A survey on tree matching and XML retrieval

Mohammed Amin Tahraoui, Karen Pinel-Sauvagnat, Cyril Laitang, Mohand Boughanem, Hamamache Kheddouci, Lei Ning

A survey on tree matching and XML retrieval[☆]

Mohammed Amin Tahraoui[a], *Karen Pinel-Sauvagnat*[b,*], *Cyril Laitang*[b], *Mohand Boughanem*[b], *Hamamache Kheddouci*[a], *Lei Ning*[c]

[a] *LIRIS, UMR5205, CNRS, Université de Lyon, Université Lyon 1, F-69622, France*
[b] *IRIT-SIG, Université Paul Sabatier, Université de Toulouse, France*
[c] *Laboratoire GAMA, Université de Lyon 1, Université de Lyon, F-69622, France*

A B S T R A C T

With the increasing number of available XML documents, numerous approaches for retrieval have been proposed in the literature. They usually use the tree representation of documents and queries to process them, whether in an implicit or explicit way. Although retrieving XML documents can be considered as a tree matching problem between the query tree and the document trees, only a few approaches take advantage of the algorithms and methods proposed by the graph theory. In this paper, we aim at studying the theoretical approaches proposed in the literature for tree matching and at seeing how these approaches have been adapted to XML querying and retrieval, from both an exact and an approximate matching perspective. This study will allow us to highlight theoretical aspects of graph theory that have not been yet explored in XML retrieval.

Contents

1. Introduction 2
   1.1. Using structure for retrieval 2
   1.2. Challenges 2
   1.3. Aim of the paper and outline 3
2. Querying XML data: key points 3
   2.1. Tree representation of XML documents 3
   2.2. XML retrieval: structured text retrieval vs querying semi-structured data 4
   2.3. Query languages 5
      2.3.1. NEXI 5
      2.3.2. XQuery 6
      2.3.3. XQuery Full-text 6

[☆] This work has received support from the French National Agency for Research (ANR) on the reference ANR-08-CORD-009.
[*] *Correspondence to:* IRIT-SIG, 118 route de Narbonne, F-31062 Toulouse Cedex 5, France. Tel.: +33 5 61 55 63 22.
E-mail addresses: Karen.Sauvagnat@irit.fr, sauvagnat@irit.fr (K. Pinel-Sauvagnat).

3. Algorithms for tree matching 6
3.1. Exact tree matching 7
3.1.1. Traversal approaches 7
3.1.2. Decomposition approaches 7
3.2. Approximate tree matching algorithms 8
3.2.1. Tree edit distance 8
3.2.2. Tree inclusion 8
3.2.3. Tree alignment distance 9
3.2.4. Summary 9
4. Tree matching and XML retrieval 10
4.1. Exact tree-matching algorithms for XML retrieval 10
4.1.1. Structural join approaches 11
4.1.2. Holistic twig join approaches 11
4.1.3. Sequence matching approaches 13
4.1.4. Other important exact XML tree algorithms 13
4.1.5. Discussion 13
4.2. Approximate tree matching algorithms for XML retrieval 14
4.2.1. Pre-processing of trees before the matching process 15
4.2.2. Approaches based on tree inclusion 16
4.2.3. Approaches based on tree edit distance 16
4.2.4. Discussion 16
5. Evaluation 17
5.1. Efficiency and effectiveness 17
5.2. The INEX evaluation campaign 17
5.2.1. The ad hoc track 17
5.2.2. The data-centric track 18
5.2.3. The efficiency track 18
6. Discussions and future research directions 19
References 19

1. Introduction

Among all available formats used to represent information (from text to video or audio), the XML (*eXtensible Markup Language*) [1] format is now extensively used. XML was designed by the W3C (*World Wide Web Consortium*) to "meet the challenges of large scale electronic publishing" and "playing an important role in the exchange of a wide variety of data on the Web and elsewhere" [2]. XML is simple and self-descriptive, and is thus used in a broad suite of Internet or digital libraries applications.

The growing number of XML documents leads to the need for appropriate **retrieval methods**, able to exploit the specific features of this type of documents. Indeed, in XML documents, textual content (data) is hierarchically structured with tags. As opposed to other markup languages (like HTML for example), tags are used to specify semantic information about the content, and not for presentation purposes. Although structure allows to organize content in XML documents, it is not always used with the same intent. XML documents can be either **data-oriented**, where structure is intensively used to organize data, and where XML components can be seen as database records, or **text-oriented**, where structure is irregular and documents are designed to be used by humans.

1.1. *Using structure for retrieval*

The hierarchical document structure can be used in the three main steps of the retrieval process [3]:

- in the querying step, to precisely define the user need (one talks about structure-oriented queries, in contrast to content-oriented ones that are only composed of keyword terms),
- in the indexing step, to index separate elements and their relations,
- in the retrieval step, to focus on the user need, i.e., to return specific required document components (i.e., documents parts) instead of entire documents to users.

Many approaches have been proposed in the literature for XML retrieval. However, most of them propose *ab nihilo* solutions, although using some concepts of graph theory could be of interest. Indeed, the underlying data model of XML documents allows to consider them as a particular kind of graphs, i.e., trees [4]. More precisely, they can be considered as labeled trees where nodes are XML elements or atomic data (i.e., text). Edges represent relations between elements. The same representation can be used for structured queries. Considering this data model for retrieval, the retrieval process can thus be seen as a matching problem between query and document trees.

1.2. *Challenges*

Many issues have to be considered by retrieval systems dealing with structured queries and the tree-matching problem.

A first problem is **efficiency**. Documents may be of very large size, and when the query is not very selective, the

```
<article year="2001">
   <header>
      <title>Information retrieval on the Web </title>
      <author>A. Dupont</author>
   </header>
   <body>
      <section>
            <title> The history of hypertext</title>
            <paragraph>In order to describe what hypertext is…</paragraph>
            <paragraph>…</paragraph>
      </section>
      <section>
            <title>Different kinds of Web Search engines </title>
            <paragraph>Plain-text search engines…</paragraph>
            <paragraph>…</paragraph>
            <paragraph>…</paragraph>
      </section>
      <section>
      …
      </section>
   </body>
<article>
```

Fig. 1 – An example of XML document.

answer set may be composed of many results. Efficient methods for storing, indexing and querying XML data are therefore required. Ranking answers may also be of interest to solve the many-answers problem.

Another challenge is **how to interpret structural constraints** [3]. The collection of XML documents may not contain documents that fit exactly the structural hints of the query. Therefore, one of the main issue to address is how to select elements that match even approximately the query constraints. For instance, a user asking for component A child of component B may be satisfied with A as descendant of B. He may also be satisfied with elements having similar tag names than the ones expressed in queries (e.g. a user looking for an element labeled *section* may be satisfied with a *sec* element).

At last, **content condition of queries, if they exist, should be combined with structural ones**. In the same way than for structural constraints, content constraints can either be strictly or loosely interpreted. In the latter case, relevance of content should be evaluated and combined with structural relevance to select the final answer set (which may or not be a ranked list).

1.3. *Aim of the paper and outline*

As aforementioned, retrieving XML documents can be considered as a tree matching problem. Although the graph theory proposes numerous algorithms for tree matching, to our knowledge, there are no surveys in the literature that examine this question with this perspective. Some surveys on XML retrieval can however be mentioned, [5,6] or some entries in the Encyclopedia of Database Systems [7]. These surveys either consider the problem of XML retrieval in a "pure" Information Retrieval point of view [5] (i.e., the focus of the survey is on ranking strategies based on information retrieval methods), or do not focus on XML retrieval but on XML similarity in general [6].

Our aim in this survey is to provide a literature review related to tree matching algorithms in a general perspective and to discuss how these algorithms have been adapted to XML retrieval purpose. We will study the matching from an exact and approximate point of view. Exact matching approaches have been mainly exploited for data-oriented documents, whilst approximate ones are mainly used for text-oriented documents. This survey will also allow us to highlight theoretical aspects that have not been yet explored and that can be of interest for XML retrieval.

The rest of the paper is structured as follows. Section 2 presents some important concepts and background concerning the querying of XML data, including query languages. Section 3 lists the most known algorithms for approximate and exact tree matching. Approaches for XML retrieval using tree matching are then presented in Section 4. Evaluation methods of these approaches are exposed in Section 5. Finally, a discussion about the interest and the impact of the presented approaches is done in Section 6, and some tracks to be explored conclude the paper.

2. Querying XML data: key points

Before presenting the algorithms proposed by graph theory for tree matching, we recall some backgrounds on XML documents and query languages, and detail issues behind the retrieval of XML information.

2.1. *Tree representation of XML documents*

In XML documents, tags are used to hierarchically and semantically organize information. In the XML document presented in Fig. 1 for example, content is organized within a *header* and a *body* tag. The *body* tag contains *section* elements, which are in turn composed of *title* and *paragraph* elements, etc.

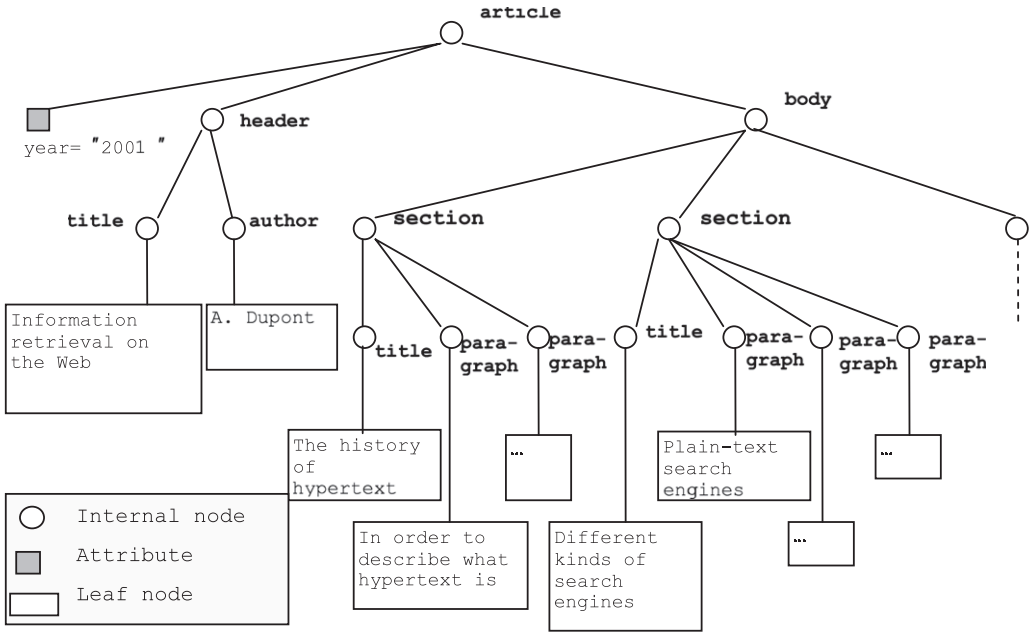

Fig. 2 – XML tree associated with the document of Fig. 1.

An element starts with an opening tag <tag> and ends with a closing tag </tag>. It may contain atomic data (as for example the *author* element in Fig. 1), other elements (for instance *section* elements in Fig. 1 contain *title* and *paragraph* elements) or a mixture of both (one talk about mixed content). Elements are also called components.

Thanks to this data model, XML documents can be represented as labeled trees [4]. In an XML tree, the whole document is represented by the **root node**, elements are represented by **internal nodes** (i.e., non terminal nodes) and the content itself is in **leaf nodes** (i.e., terminal nodes). These nodes are linked with edges showing their hierarchical relations. The tree representation of the document in Fig. 1 is given in Fig. 2.

According to the type of their content, XML documents can be categorized into two groups: **data-oriented documents** or **text-oriented documents**.

Documents of the first category have a fine granularity, are highly structured and do not contain mixed contents. The order of children in a given element is often without any importance. Elements can be considered as database records (i.e. like couples of key-value), and information content is often small. The document in Fig. 3 is an example of a data-oriented document.

On the other hand, text-oriented documents are often loosely structured: they have an irregular structure and may contain many mixed contents. They are designed to be used and read by humans. Books or scientific articles are good examples of this type of documents. Moreover, the order of elements is very important to understand the document content. For example, in the text-oriented document of Fig. 1, *section* elements should be read in the good order to make the whole article understandable.

2.2. *XML retrieval: structured text retrieval vs querying semi-structured data*

In the literature, XML document access has been studied according to two different points of view:

- the *database* one, which considers exact matching of documents and queries and which focuses on efficiency problems. Approaches are more concerned with searching than ranking.
- the *information retrieval* one, which considers approximate matching of documents and queries and which aims at ranking document components according to their relevance to the query.

Historically, highly structured documents were considered by the database community (which talks about *querying semi-structured data*), whereas the Information Retrieval (IR) community proposed approaches for searching in text-oriented documents (this is what the community calls *structured text retrieval*).

Boundaries between the two types of approaches are however nowadays not so strict. As stated by Lalmas and Baeza-Yates in [8]:

> "From a terminology point of view, structured text retrieval and querying semi-structured data, in terms of end goals, are the same, i.e., finding elements answering a given query. The difference comes from the fact that in information retrieval, the structure is added, and in databases, the structure is loosened".

In the following, we will thus describe some approaches from the database community (exact matching, Section 4.1) and from the information retrieval community (approximate matching, Section 4.2). In this last case, the set-up of INEX (*Initiative for the Evaluation of XML Retrieval*) in 2002 [9]

```
<publications>
   <inproceedings>
      <title>Information retrieval on the Web </title>
      <author>A. Dupont</author>
      <year>2001</year>
      <conference>WWW Conference</conference>
   </inproceedings>
   <inproceedings>
      <author>J. Allan</author>
      <title>Information retrieval and graphs</title>
      <conference>AOC 2008</conference>
      <year>2008</year>
   </inproceedings>
   …
<publications>
```

Fig. 3 – Example of data-oriented document.

promoted the development of many proposals. Most of the approaches presented here were thus experimented in this framework.

Whatever the considered approach, the problem is to match a document tree with a query tree. The following paragraph presents the different query languages proposed in the literature for XML retrieval.

2.3. *Query languages*

Queries for XML retrieval can be classified into *content-only* and *content-and-structure queries*.

Content-only queries are composed of simple keywords terms, and have historically been used in traditional information retrieval. They are also suitable for XML retrieval, in retrieval scenarios where the user do not know the structure of documents he/she is querying. In this paper, we are not interested in such queries, because they do not have structural hints. The main problem for such queries is to find the good granularity of information to be returned to users, and not to match documents and queries trees.

In content and structure queries, users provide content conditions linked with structure hints. They are two reasons that lead a user to add structural constraints to a query [10]:

- the first one is to reduce the size of the result,
- the second one is to restrict the search to document parts that are supposed to be relevant.

According to Amer-Yahia and Lalmas [11], there are three main categories of content and structure query languages:

- *tag-based* queries allow users to express very simple conditions concerning the tag of the element in which the required content should be. They are of the form: "tag: content condition". For example the query "*section: search engines*" means that the user is looking for a section element about "search engines".
- *path-based* queries are based on the XPath [12] syntax. They include content conditions in a XPath-based syntax. Examples of languages allowing path-based queries are the NEXI language [13] or FuzzyXPath [14].

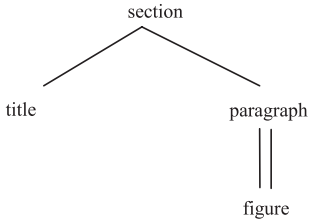

Fig. 4 – A twig query.

- *clause-based* queries have a structure similar to the one of SQL. They contain nested clauses that allow to express the used need. In XML retrieval, XQuery [15] or XQuery full-text [16] are examples of clause-based queries.

If we purely consider the structure conditions of queries, XQuery and XPath [15,12] queries can be basically divided into two main groups: path queries and twig queries. Path queries are simple queries which contain path expressions, i.e., child axis "/" and descendant axis "//". Twig queries are represented as node-labeled twig patterns, i.e., small trees. They are also named tree pattern queries.

Fig. 4 illustrates an example of twig query.

Since 2000, a lot of algorithms [17–28] have been proposed to process twig queries in an exact matching point of view. A short survey of these algorithms is presented in Section 4.1.

We describe in the following section three representative languages for the families they belong to: NEXI, XQuery and XQuery full-text.

2.3.1. *NEXI*

The NEXI query language was introduced in the context of the INEX evaluation campaign in 2004 [13]. NEXI is a subset of XPath, where an *about* function is added to express content conditions. Authors argue that this is the simplest query language that could possibly work.

Two types of queries can be expressed in NEXI: CO (*Content Only*) and CAS (*Content and Structure*) Queries. As CO queries are composed of simple keywords terms, they will not be considered here. An example of CAS query can be:

```
//article[about(.//references, information retrieval)] //
 section[about(.,search engines)]
```

This twig query means that the user is looking for a *section* (which is called target element) about "search engines" contained in an *article* (called support element) which should contain references about "information retrieval". The *about* function contains a content condition, expressed as a CO query ("information retrieval" or "search engines" in our example).

The NEXI query language has been extended for question answering [29], searching in heterogeneous collections (i.e., collections containing documents having a different DTD) and multimedia retrieval [30].

2.3.2. *XQuery*

Historically, before the W3C recommendation for XQuery [15], many query languages have been proposed in the literature for searching in XML documents. We can cite Un-QL [31], XML-QL [32], XQL [33] or Quilt [34]. XQuery is inspired from all of them.

XQuery queries are FLWOR (For Let Where Order by Return) expressions. The *for* clause defines one or more variables and iterates over elements resulting of an expression. The *let* clause also defines one or more variables, but does not iterate on them. The optional *where* clause expresses a selection condition. At last, before constructing the result in the *return* clause, the optional *order by* clause can be used to specify a list of sorting criteria. An example of XQuery query can be found in [35]:

```
<bib>
{
  for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year}">
    { $b/title }
    </book>
 }
</bib>
```

This query finds books published after 1991 by Addison-Wesley. Results are <book> elements containing <title> elements of the corresponding resulting books and having the publication date as attribute.

2.3.3. *XQuery Full-text*

XQuery Full-text [16] is a proposition of the W3C to extend the XQuery language to full-text search. Full-Text-oriented functionalities of XQuery Full-text are for example token and phrase search, token ordering or token proximity.

One of the most important feature of the language is that it allows to rank results by relevance, and that the user can define its own relevance. This is also possible to use tools such as stemming, thesauri, stop-words or regular expressions [36].

As example, let us consider the following Xquery full-text query, taken from [37]:

```
for $book in doc("http://bstore1.example.com/full-text.xml")
   /books/book
let $booktext := $book/content [. contains text ("conduct"
   ftand "usability" ftand "tests" distance at most
   10 words) using stemming]
let score $s := $booktext contains text
   (("measuring" ftand "success" distance
   at most 4 words) weight 1.8) using stemming
where $booktext
order by $s descending
return ($book/metadata/title, $booktext)
```

This query finds books which discuss "conducting usability tests". Those mentioning "measuring success" are returned first. The following features are used: stemming, unordered distance (no more than 10 words between "conduct" and "usability" and between "usability" and "tests" are for example required), and weight declaration on optional words to impact the scoring and sorting.

As seen in this section, there are a lot of possible languages to search in XML documents collection. As stated in [5], the complexity and expressiveness of these languages increase from tag-based to clause-based queries. All languages have their own properties and potential uses. They all need a learning phase to be able to use them, but with different degree of ease. For example, the information retrieval-oriented NEXI language is introduced as "the simplest query language that could possibly work" [13], whereas the database-oriented XQuery Full-text query language propose a very complete but hard to learn syntax for end-user.

To overcome this problem of complexity for end users, graphical query languages were proposed [38,39], but they are far from being extensively used.

Whatever the query language used, content and structure queries, in the same manner than XML documents, can be represented as labeled trees. The retrieval process can thus be summarized to a tree matching process. The following section describes state-of-the-art algorithms for exact and approximate tree matching.

3. Algorithms for tree matching

In order to state the problem of tree matching, we first define some concepts related to labeled tree and their components.

A labeled tree is denoted as $T = (V, E, r, \mu)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a finite set of nodes, $E = \{(u, v) | u, v \in V\}$ is a set of edges. $r \in V$ is a distinguished node called the *root*, and μ is a labeling function which maps each node of T to one of labels in a finite set $L = \{l_1, l_2, \ldots, l_k\}$. For simplicity, in the remaining of this paper, we call a rooted labeled tree simply as a tree. We define the level $l(u)$ of node u as the number of edges along the unique path between itself and the root. The *depth* of a rooted tree is the maximum level of any vertex in the tree. Given two adjacent nodes u and v in a rooted tree, with $l(v) > l(u)$, v is called the *child* of u, conversely, u is a *parent* of v. A node of T without children is called a *leaf*, and the other nodes are called *internal nodes*. Nodes having the same parent are called *siblings*. Finally, an ordered tree is a rooted tree for which an ordering is specified for the children of every node, and it is unordered otherwise.

Throughout this section, we use the following notations:

- T_1 and T_2 are two rooted trees to be matched,
- d_i, l_i and $\triangle_i$ respectively denote the depth, the number of leaves and the maximum degree of T_i (i.e., the maximum

degree of its vertices, where the degree of a vertex is the number of edges incident to the vertex), where $i = 1, 2$.

This section reviews the exact and approximate tree matching algorithms. In the first part, we give a brief overview of algorithmic results in exact tree matching. Most attention in the literature has been devoted to approximate tree matching which is described in Section 3.2.

3.1. *Exact tree matching*

We can define the exact tree matching problem as follows. Let target $T_1 = (V_1, E_1, r_1, \mu_1)$ and pattern $P = T_2 = (V_2, E_2, r_2, \mu_2)$ be two ordered labeled trees. T_2 matches T_1 at node r_1 if there exists a mapping from nodes of T_2 into the nodes of T_1 such that:

1. the root of T_2 maps to r_1,
2. if $v_2 \in V_2$ maps to $v_1 \in V_1$, then $\mu_1(v_1) = \mu_2(v_2)$,
3. if $v_2 \in V_2$ maps to $v_1 \in V_1$ and v_1 is not a leaf, then each child of v_2 maps to some child of v_1.

Fig. 5 shows an example of tree pattern matching between pattern tree P and target tree T, where the one-to-one mappings are represented by dotted lines. The tree pattern matching problem has been extensively studied and has many important applications including term rewriting systems, transformational programming systems, code generator–generators and a variety of functional languages with equational function definitions (see [40]). The following is a brief overview of general exact tree pattern matching algorithms.

Given tree pattern P and target T of size m and n respectively where $m \leq n$, a native tree pattern matching algorithm takes $O(mn)$ in the worst case. The basic idea of this algorithm consists in visiting all vertices of T in pre-order walk. For each visited vertex v, the algorithm calls a special recursive procedure to test for possible occurrence of P at the vertex v. The recursive procedure is terminated when a mismatch is detected.

Several attempts were made to improve the naive $O(mn)$ step algorithm. These attempts can be classified into two general categories: namely *traversal approaches* and *decomposition approaches*.

3.1.1. *Traversal approaches*

In [40], Hoffmann and O'Donnell proposed two traversing methods, called *bottom-up* and *top-down* matching algorithms with the same worst case bound, where *bottom-up* algorithm traces the tree from the leaves to the root and *top-down* algorithm works by traversing the tree from the root to leaves. *Top-down* algorithm is efficient in space, but in non-linear time. The main idea of this algorithm is to encode the root-to-leaf paths as strings. It then finds all occurrences of each string in the target tree according to the string pattern matching algorithm of Aho and Corasick [41]. On the other hand, the key idea of the *bottom-up* technique is to find, for each node in the target tree, all patterns and all parts of patterns that match this node. *Bottom-up* algorithm performs in linear time. However, extra storage space is needed during matching to store and establish the table encoding of all nodes in the target tree.

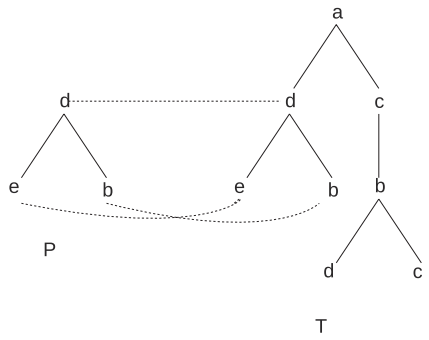

Fig. 5 – An example of tree pattern matching.

Hoffmann and O'Donnell stimulated a number of additional studies offering heuristic for space improvements. Chase method [42] have received considerable attention in the literature. This method was thought to improve *bottom-up* solutions presented by Hoffmann and O'Donnell using the deeper structure of the pattern set P to reduce the computational cost of bottom-up algorithm. Chase proved that this transition map better utilizes space than Hoffmann and O'Donnells. Cai et al. [43] improved the preprocessing time and space complexities of Chase *bottom-up* algorithm. The main modification introduced in Chase method is that the preprocessing can be done incrementally with respect to additions and deletions of pattern tree.

3.1.2. *Decomposition approaches*

The basic idea of these approaches is to decompose the tree into small pieces in order to facilitate the matching process. In [44], Burghardt proposed a tree pattern algorithm, which consists of two phases. In the first phase a matching automaton is constructed from a given pattern set. For this, it generalizes the string matching algorithm of Aho and Corasick [41] such that each pattern is decomposed depending on the position ordering relation. Then, these patterns are merged into an automaton. In the second phase, one or more target trees are fed into the automaton, by traversing through the automaton according to the target trees.

Kosaraju [45] proposed a new algorithm with improved complexity bound from $O(nm)$ to $(nm^{\frac{3}{4}})$. This algorithm is based on three new techniques: suffix tree of a tree, the convolution of a tree and a string, and partitioning of trees into chains and anti-chains. Dubiner et al. [46] improved Kosaraju's algorithm by discovering and exploiting periodical strings appearing in the pattern. They obtained a bound of $(nm^{\frac{1}{2}})$. Cole and Hariharan [47] introduced and proposed an efficient algorithm for the *subset Matching problem*. The goal of this problem is to report all occurrences of a pattern string P of length m in a string T of length n. Then the tree pattern matching has been reduced to a number of smaller subset matching problem. Combining this reduction with the subset matching algorithm take an $o(n \log^3 m)$ time randomized algorithm for the tree pattern matching problem. Later the same authors improved the reduction time complexity [48]. They obtained an $O(n \log^2 m + m)$ time deterministic algorithm and an $O(n \log n + m)$ time Monte Carlo algorithm for the tree pattern matching problem.

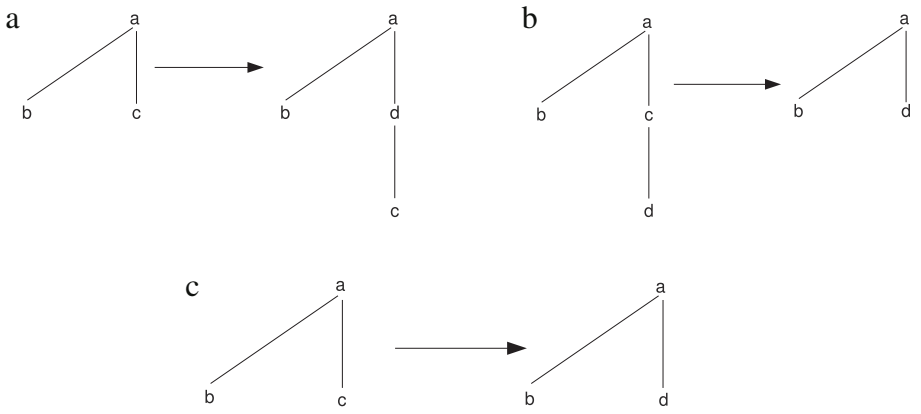

Fig. 6 – Tree edit distance operations: (a) node insertion (b) node deletion (c) node relabeling.

3.2. *Approximate tree matching algorithms*

Approximate tree matching is the process of determining the best possible match of one tree against another one. There are two types of approximate tree matching: the unordered tree matching problem and the ordered tree matching problem. The tree matching in these two types refers to the task of measuring the dissimilarity between trees. One of the most known methods for evaluating the dissimilarity is the edit distance metric. In a nutshell, the tree-edit distance metric is the natural generalization of the well-known string edit distance problem. The similarity of edit-distance between two trees T_1 and T_2 is defined as the minimum cost sequence of basic edit operations required to transform T_1 into T_2. There are three basic edit operations: *node insertion*, *node deletion* and *node relabeling*. We describe each of these operations in detail below.

1. *Node insertion:* insert node d as a child of parent a in T, making d the parent of some of the children of a, as shown in Fig. 6(a).
2. *Node deletion:* inverse of insert. Delete a non-root node c of T with parent a, making the children of c become the children of a. The children are inserted in the place of c into the set of children of a (Fig. 6(b)).
3. *Node relabeling:* replace a label of a node by another label, as shown in Fig. 6(c).

Based on the concept of tree edit operations, Bille [49] derived three main groups of algorithms for tree matching: *tree edit distance approaches*, *tree inclusion* and the *tree alignment distance*. The ordered tree inclusion problem has been recognized as an important query primitive in XML databases [50–52], since an XML document can be viewed as a rooted, ordered and labeled tree. In the rest of this section, we extend the survey of Bille [49] by adding the current trends in approximate ordered tree matching algorithms.

3.2.1. *Tree edit distance*

The tree edit distance between two trees T_1 and T_2 is defined as the minimum cost sequence of edit operations that turns T_1 into T_2. In the literature, Tai [53] was the first to propose a recursive algorithm for evaluating the edit distance between two given ordered labeled trees T_1 and T_2. The resulting algorithm has a complexity of $O(|T_1| \cdot d_1^2 \cdot |T_2| \cdot d_2^2)$ time and space. This algorithm was improved by Zhang and Shasha [54]. It runs in $O(|T_1| \cdot |T_2| \cdot min\{l_1, d_1\} \cdot \min\{l_2, d_2\})$ time and $O(|T_1| \cdot |T_2|)$ space. In [55], Klein developed a faster algorithm which has better time complexity in the worst case. In this algorithm, the tree edit distance problem can be solved in $O(|T_1|^2 \cdot |T_2| \cdot \log|T_2|)$ time and $O(|T_1| \cdot |T_2|)$ space. Chen [56] presented a new bottom-up recurrent approach which runs in $O(|T_1| \cdot |T_2| + l_1^2 \cdot |T_2| + l_1^{\frac{5}{2}} \cdot l_2)$ time and $O((|T_1| + l_1^2) \cdot \min(l_2, d_2) + |T_2|)$ space.

Dulucq and Touzet [57] studied the behavior of dynamic programming methods given in [55,54]. They showed that those two algorithms can be described as decomposition strategies. They then introduced the definition of cover strategies, that are natural and easy to handle extensions of Klein and Zhang–Shasha algorithms. In this framework, a novel algorithm has been provided to minimize the number of distinct recursive calls. By construction, this algorithm involves less relevant forests than Zhang–Shasha and Klein strategies. So, it is at most in $O(|T_1| \cdot |T_2| \cdot \log|T_1| \cdot \log|T_2|)$ time complexity in the worst case, and in $O(|T_1| \cdot |T_2|)$ in average. Demaine et al. [58] proposed a new algorithm for computing the tree edit distance that falls into the same decomposition strategy framework of [57,55,54]. In the worst-case, this algorithm requires $O(|T_1| \cdot |T_2| \cdot (1 + \log \frac{|T_1|}{|T_2|}))$ time and $O(|T_1| \cdot |T_2|)$ space.

Many other solutions have been developed, see [49] for survey. Among all these algorithms, Klein algorithm and the decomposition strategy are the fastest in terms of worst case time and space complexity.

3.2.2. *Tree inclusion*

Given two labeled rooted trees P and T, the tree inclusion problem is to locate the subtrees of T that are instances of P. We can see that the tree inclusion is a particular case of the tree edit distance problem. In fact, the tree pattern P is included in T if we can obtain P from T by a sequence of delete operations. Fig. 7 shows an example of tree inclusion such that the tree P is included in the tree T by deleting the nodes labeled d, e, f and g.

The tree inclusion problem was originally posed by Knuth [59]. There are two types of inclusion problems: the

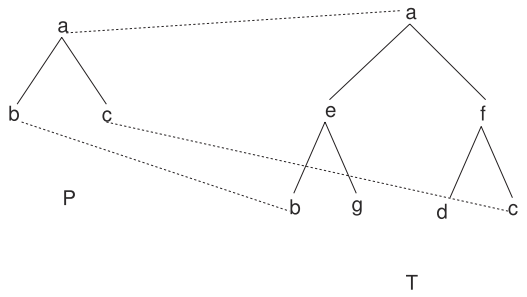

Fig. 7 – An example of tree inclusion matching.

unordered tree inclusion problem, where the order among siblings of the pattern nodes is not important, and the ordered tree inclusion problem, where the left-to-right order among siblings is important. Kilpelainen and Mannila [60] studied the two versions of this problem. They showed that this kind of problem is NP-complete for unordered tree.

An injective function f from the nodes of P to the nodes of T is an embedding of P into T, if it preserves labels and ancestorship. For every pair of nodes u, v in P, we require that:

1. $f(u) = f(v)$ if and only if $u = v$.
2. the two vertices u and $f(u)$ have the same label.
3. u is an ancestor of v in P if and only if $f(u)$ is an ancestor of $f(v)$ in T.

We say that pattern P is an included tree of T, and T is an including tree of P, if there is an embedding of P into T.

The ordered tree inclusion problem results from the unordered tree inclusion by fixing the left-to-right order of nodes. Indeed, an embedding of a tree P into a tree T is an ordered embedding of P into T if and only if the sibling conditions are hold *i.e.*, the vertex v is the left of u in P iff $f(v)$ is the left of $f(u)$ in T.

With dynamic programming method, Kilpelainen and Mannila [60] proposed the first polynomial tree inclusion algorithm using $O(|P| \cdot |T|)$ time and space. Most of the later improvements are essentially based on a dynamic programming technique from the original algorithm of [60]. The basic idea behind this algorithm is the following: for all pairs $(v, u) \in V(P) \times V(T)$, let $v_1, v_2, \ldots, v_p$ be the children of v and $u_1, u_2, \ldots, u_q$ be the children of u. To decide if the subtree rooted at a node v can be included in the subtree rooted at a node u, we try to find a sequence of numbers $1 \le x_1 \le x_2 \le \cdots \le x_q \le q$ such that the subtree rooted at a node v_k can be included in the subtree rooted at a node u_{xk} for all k, $1 \le k \le p$.

Richter [61] proposed a new algorithm using $O(|\sum_P| \cdot |T| + m(P,T) \cdot d_T)$ time, where $\sum_P$ is the set of the labels of the tree pattern and $m(P,T)$ is the number of pairs $(v,u) \in V(P) \times V(T)$ with $Label(v) = Label(u)$. The space complexity of this algorithm is $O(|\sum_P| \cdot |T| + m(P,T))$. Hence, if the number of matches is small, the time complexity of this algorithm is better than $O(|T| \cdot |P|)$.

In [62], a more complex algorithm was presented using $O(|T| \cdot l_P)$ time and $O(l_P \cdot min\{d_T, l_T\})$ space. In [63], a *bottom-up* algorithm is proposed. The main idea of this algorithm is to construct a data structure on T supporting a small number of procedures. Then three algorithms are proposed for each data structure, the combining of these three algorithms give a time complexity bounded by the minimum of the following three values:

1. $\frac{|P| \cdot |T|}{\log(|T|)}$.
2. $l_P \cdot |T| \cdot \log\log(|T|)$.
3. $l_P \cdot |T| \cdot \log\log(|T|)$.

Moreover, the space of this algorithm is improved by a linear factor $O(|T| + |P|)$. This will make the algorithm more efficient and faster to query a large XML document. Chen and Chen [64] proposed a new tree inclusion algorithm, which requires $O(|T| \cdot \min\{d_P, l_P\})$ time and $O(|T| + |P|)$ space. However, there are flaws in the time complexity analysis of this algorithm. These flaws are shown by two counterexamples presented in [65]. Hence the time complexity of Chen and Chen algorithm is not polynomial. More recently, Chen and Chen [66] revisited their work and present a new *top-down* algorithm to remove any redundancy of [65]. The time complexity of the new one is bounded by $O(|T| \cdot \min\{d_P, l_P\})$. The space requirement remains with $O(|T| + |P|)$.

3.2.3. *Tree alignment distance*

The tree alignment problem was introduced as natural generalizations of string edit distance [67]. It is a special case of the tree editing problem, where all insertion operations must be done before any deletions. An alignment AL between two labeled trees T_1 and T_2 is obtained by first performing insert operations of nodes labeled with the null symbol λ (called space) on the two trees so they become isomorphic, and then overlaying the first augmented tree on the second. A cost is defined for each pair of opposing labels in AL. The cost of AL is the sum of costs of all opposing labels in AL. The matching problem here is to find an alignment with the minimum cost. Fig. 8 shows an example (from [68]) of an ordered alignment.

Jiang et al. [68] were the first to propose an ordered tree alignment algorithm, that studies algorithms for both ordered and unordered version of this problem. Kuboyama et al. [69] showed that the alignment problem for two unordered trees has no polynomial time absolute approximation algorithm, unless P = NP. We survey here the main algorithms of ordered tree alignment problem. The algorithm of Jiang et al. [68] uses $O(|T_1| \cdot |T_2| \cdot (\triangle_{T_1} + \triangle_{T_2})^2)$ time and space. Hence, for small degree trees, this algorithm is in general less expressive than the best known algorithm for the edit distance. Jansson and Lingas [70] presented a fast algorithm for ordered alignment between two similar trees. This algorithm can construct an optimal alignment of T_1 and T_2 using at most k spaces in $O((|T_1| + |T_2|) \cdot \log(|T_1| + |T_2|) \cdot (\triangle_{T_1} + \triangle_{T_2})^4 \cdot k^2)$ time. Wang and Zhao [71] proposed a new ordered tree alignment algorithm which is based on the constrained edit distance. This algorithm trees runs in $O(|T_1| \cdot |T_2| (\triangle_{T_1} + \triangle_{T_2})^2)$ time and requires $O(\log(|T_1|) \cdot |T_2| \cdot (\triangle_{T_1} + \triangle_{T_2}) \cdot \triangle_{T_1})$ space.

3.2.4. *Summary*

Table 1 recalls the time and space complexities of the algorithms covered in this section.

To summarize, the problem of tree matching has been well studied in the literature. We provided in this section an overview of important algorithms. Due to the structure of XML documents, it seems obvious and natural to use those tree matching algorithms for XML retrieval. However, most

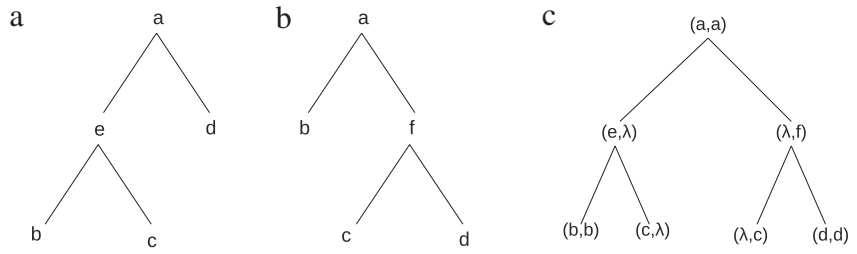

Fig. 8 – Tree Alignment Distance: (a) tree T_1, (b) tree T_2, (c) an alignment of T_1 and T_2.

Table 1 – The most important approximate ordered tree matching algorithms and their complexities.

Algorithm	Time	Space
Tree edit distance		
Tai [53]	$O(\|T_1\| \cdot \|T_2\| \cdot d_1^2 \cdot d_2^2)$	$O(\|T_1\| \cdot \|T_2\| \cdot d_1^2 \cdot d_2^2)$
Zhang and Shasha [54]	$O(\|T_1\| \cdot \|T_2\| \cdot \min\{d_1, l_1\} \cdot \min\{d_2, l_2\})$	$O(\|T_1\| \cdot \|T_2\|)$
Klein [55]	$O(\|T_1\|^2 \cdot \|T_2\| \cdot \log \|T_2\|)$	$O(\|T_1\| \cdot \|T_2\|)$
Dulucq and Touzet [57]	$O(\|T_1\| \cdot \|T_2\| \cdot \log \|T_1\| \cdot \log \|T_2\|)$	$O(\|T_1\| \cdot \|T_2\|)$
Demaine et al. [58]	$O\left(\|T_1\| \cdot \|T_2\| \cdot \left(1 + \log \frac{\|T_1\|}{\|T_2\|}\right)\right)$	$O(\|T_1\| \cdot \|T_2\|)$
Chen [56]	$O(\|T_1\| \cdot \|T_2\| + l_1^2 \cdot \|T_2\| + l_1^{\overset{>}{2}} \cdot l_2)$	$O((\|T_1\| + l_1^2) \cdot \min(l_2, d_2) + \|T_2\|)$
Tree inclusion problem		
For unordered trees [60]	NP-Complete	
Kilpelainen and Mannila [60]	$O(\|P\| \cdot \|T\|)$	$O(\|P\| \cdot \|T\|)$
Richter [61]	$O(\sum_P \cdot \|T\| + m(P, T) \cdot d_T)$	$O(\sum_P \cdot \|T\| + m(P, T))$
Chen [62]	$O(\|T\| \cdot l_P)$	$O(l_P \cdot \min\{d_T, l_T\})$
Bille and Gortz [63]	$\min\left\{\frac{\|P\| \cdot \|T\|}{\log \|T\|}, l_P \cdot \|T\| \cdot \log\log(\|T\|), l_P \cdot \|T\| \cdot \log\log(\|T\|)\right\}$	$O(\|T\| + \|P\|)$
Chen and Chen [64]	flaws in the time complexity computation	$O(\|T\| + \|P\|)$
Chen and Chen [66]	$O(\|T\| \cdot \min\{d_P, l_P\})$	$O(\|T\| + \|P\|)$
Tree alignment distance		
Jiang et al. [68]	$O(\|T_1\| \cdot \|T_2\| \cdot (\triangle_{T_1} + \triangle_{T_2})^2)$	$O(\|T_1\| \cdot \|T_2\| \cdot (\triangle_{T_1} + \triangle_{T_2})^2)$
Jansson and Lingas [70]	$O((\|T_1\| + \|T_2\|) \cdot \log(\|T_1\| + \|T_2\|) \cdot (\triangle_{T_1} + \triangle_{T_2})^4 \cdot k^2)$	$O((\|T_1\| + \|T_2\|) \cdot \log(\|T_1\| + \|T_2\|) \cdot (\triangle_{T_1} + \triangle_{T_2})^4 \cdot k^2)$
Wang and Zhao [71]	$O(\|T_1\| \cdot \|T_2\| (\triangle_{T_1} + \triangle_{T_2})^2)$	$O(\log(\|T_1\|) \cdot \|T_2\| \cdot (\triangle_{T_1} + \triangle_{T_2}) \cdot \triangle_{T_1})$

of the state-of-art algorithms are given for general trees. They cannot be directly used on specific trees as for instance XML documents. In order to obtain an effective and efficient algorithm for XML query processing, we need to take into account the properties and the specificity of the XML trees. In fact, there are different requirements that lead to various XML tree matching algorithms different than the traditional tree matching problem presented in this section:

- most of the existing tree matching algorithms rely on the assumption that the target tree can be held entirely in main memory, which is not valid for XML tree algorithms. The latter mainly operate on minimizing access to the input data tree when performing matching operations.
- most exact matching algorithms attempt to find one matching between the pattern tree and target tree, while exact XML tree matching require the output of all possible matching which is a core operation in XML query processing.
- the XML tree pattern matching becomes more complex than the traditional tree matching due to the existence of ancestor–descendant edges in the pattern tree.

4. Tree matching and XML retrieval

We provide in this section a survey of current development of tree matching algorithms related to XML retrieval. Its content is divided into two main parts. The first covers the exact algorithms for finding all twig patterns in an XML database. The second part describes and illustrates in detail the approximate algorithms. Note that a significant number of exact tree matching improvements have been proposed since the proposition of Zhang et al. [17]. This explains that Section 4.1 is much larger than Section 4.2.

Table 2 summarizes the XML tree matching approaches presented in this section regarding the type of collection on which they are applied and the tree matching algorithm they use.

4.1. *Exact tree-matching algorithms for XML retrieval*

As we mentioned in Section 2, XML uses a tree-structured model for representing data and twig patterns for expressing queries. Finding all occurrences of such a twig pattern

Table 2 – Summary of XML approaches presented in Section 4.

Type of matching	Input: Query	Input: collection	Name of the tree matching algorithm used	Adaptation-similar approaches
Exact	Pure structure query	Data-oriented	Decomposition approaches [44–47]	**Binary decomposition** (structural join approaches): [17,18]. **Root-to-leaf paths decomposition** (holistic twig join approaches): [19,20,72–75]. **String decomposition** (sequence matching approaches): [26,76,77].
			Traversal approaches: [40,42,43]	One-phase holistic twig matching approaches: [22,78,79,23].
Approximate	Content and structure query	Text-oriented	Tree edit distance [53,57,58]	Edit distance [80], Strict use of the algorithm [81], Tree summaries [82], Relaxation [83], Optimal cover strategy [84].
			Tree inclusion [59–61,66]	[52,50]
		Data-oriented	Tree edit distance [53,57,58]	Relaxation [85]

in an XML collection is clearly a core operation in XML query processing. There has been a great deal of interest in recent years to overcome efficiently this problem. Existing XML twig pattern algorithms can be classified into two-phase algorithms (decomposition approaches) and one-phase algorithms (traversal approaches). However, most of the proposed approaches in the literature are interested in structural properties of queries and can be classified into four groups as follows:

- structural join approaches,
- holistic twig join approaches,
- sequence matching approaches,
- other important exact XML tree algorithms.

In the following, we present an overview of the main algorithms and results of each group.

4.1.1. *Structural join approaches*

In this section, we review the join based approach, a very important native idea, which usually includes three parts: (1) decomposition, (2) matching and (3) merging. Firstly, a twig pattern is decomposed into a set of basic parent–child and ancestor–descendant relationships between pairs of nodes. In the second phase, each binary relationship is separately executed using structural join techniques and its intermediate results are stored for further processing. The final result is formed by merging these intermediate results.

Indexing and node encoding is critical for the structural join approaches. The most common is the containment labeling scheme [17]. This labeling scheme (called region encoding) uses a region code $(start, end, level)$ to represent the position of an XML element in the data tree where *start* and *end* are generated by performing a pre-order traversal procedure of the data tree; *level* is the nesting depth of the element in the data tree. The positional information which follows nodes must allow decision of ancestor–descendant and parent–child relationships. Based on this labeling scheme, Zhang et al. [17] proposed the multi-predicate merge join (*MPMGJN*) algorithm, which is the first structural join to find all occurrences of the basic structural relationships. This mechanism is an extension of the classical merge-join algorithm developed in relational query optimizers for equi-joins. The results in Zhang showed that for many XML queries, MPMGJN is more than an order-of-magnitude faster than the standard Relational Database Management System (RDBMS) join implementations. Al-Khalifa et al. [18] took advantage of the same labeling scheme of XML elements to devise I/O and CPU optimal join algorithms for matching binary structural relations against an XML document.

The main problem of the approaches mentioned above is that they may generate large and possibly unnecessary intermediate results, since the join results of individual binary relationships may not appear in the final results.

4.1.2. *Holistic twig join approaches*

Until now, the holistic twig join approach was regarded as the most efficient family in the literature. Bruno et al. [19] proposed the first holistic XML twig pattern matching algorithm to avoid producing large intermediate results. This approach constituted the major attempt for several subsequent works in order to make the twig pattern matching very efficient. In the following, we present an overview of the basic ideas and results of the main holistic twig join algorithms available in the literature.

TwigStack algorithm

The main disadvantage of decomposing twig queries into multiple binary relationships is that this approach generates a large amount of intermediate query results even when the input and output size are more manageable. Frequently, such intermediate results cannot be held in main memory and must be stored on disk. This will result in high disk I/O cost. In order to overcome this weakness, Bruno et al. [19] proposed a novel holistic twig join algorithm *TwigStack*, wherein, no large intermediate results are created.

The central idea behind this approach is to use a chain of linked stacks to compactly represent intermediate results of individual query root-to-leaf paths, which are then composed to produce the final solutions. *TwigStack* avoids storing intermediate results unless they contribute to the final results when the query twig has only ancestor–descendant edges. The analytical results of this approach demonstrate that *TwigStack* is I/O and CPU optimal among all sequential algorithms that read the entire input. These analysis are

confirmed by experimental results on a range of real synthetic data and query twig patterns.

Improvements on TwigStack

The idea of holistic twig join has been adopted in several works in order to make the structural join algorithm very efficient. This section is devoted to a structured review of these advances.

Improvement 1: efficient processing of parent–child edge query. As mentioned before, when all edges in query patterns are ancestor–descendant ones, *TwigStack* ensures that each root-to-leaf intermediate solution contribute to the final results. However, this algorithm still cannot control a large number of intermediate results for parent–child edge query. A first improvement that can be found in the literature concerns the efficient handling of twig queries with parent–child relationships. Among the proposed approaches, Lu et al. [20] extended *TwigStack* by proposing *TwigStackList* algorithm. This algorithm has the same performance than *TwigStack* for query patterns with only ancestor–descendant edges, but also produces much less useless intermediate solutions than *TwigStack* for queries with parent–child relationships. The main technique of *TwigStackList* algorithm is to read more elements in the input streams and cache some of them (only those that might contribute to final answers) into lists in the main memory, so that we can make a more accurate decision to determine whether an element can contribute to the final solution or not. Chen et al. [72] suggested another algorithm, called *iTwigJoin*, which can be used on various data streaming strategies (e.g. *Tag+Level* streaming and *Prefix Path Streaming*). *Tag+Level streaming* can be optimal for both ancestor–descendant and parent–child only twig patterns whereas *Prefix Path streaming* could be optimal for ancestor–descendant only, parent–child only and one branch node only twig patterns assuming there was no repetitive tag in the twig patterns.

Improvement 2: eliminating redundant computations. *TwigStack* works by recursively calling a method called *getNext* to efficiently filter useless elements in order to return the next node for processing. The existing holistic twig join algorithms may perform many redundant checks in the call of *getNext*. Thus, an improvement strategy consists in avoiding these unnecessary computations. *TSGeneric+* [73] makes improvements on *TwigStack* by using *XR-Tree* to effectively skip some useless elements that do not contribute to the final results. The motivation to use *XR-tree* is that, the ancestors (or descendants) of any XML element indexed by an *XR-tree* can be derived with optimal worst case I/O cost. However, *TSGeneric+* may still output many useless intermediate path solutions like *TwigStack* for queries with parent–child relationship. Guoliang et al. [74] proposed the *TJEssential* algorithm based on three optimization rules to avoid some unnecessary computations. They presented two algorithms incorporated with these optimization rules to effectively answer twig patterns in leaf-to-root combining with root-to-leaf way. A novel holistic twig join algorithm, called *TwigStack+* is proposed in [86]. It is based on holistic twig join guided by extended solution extension to avoid many redundant computations in the call of *getNext*. It significantly improves the query processing cost, simply because it can check whether other elements can be processed together with current one.

Improvement 3: eliminating the merging phase. Another improvement consists in reducing the cost of queries execution. Indeed there exists very interesting approaches that eliminate the second phase of merging of individual solutions [22,23, 78,79]. These algorithms yield no intermediate results. Chen et al. [22] proposed the first tree pattern solution, called *Twig2Stack* that avoids any post path join. *Twig2Stack* is based on a hierarchical stack encoding scheme to compactly represent the twig results. The main idea is to organize the elements matching the same query node in a hierarchical structure in order to capture their A–D relationships. However, maintaining the hierarchical structure among stacks in *Twig2Stack* algorithm has a critical impact on the performance processing of a twig query. Aiming to avoid this complex hierarchical-stacks, Qin et al. [23] proposed *TwigList* algorithm. This is a simplification of *Twig2Stack* using simple lists and intervals given by pointers, which improves performance in practice. In the same context a novel algorithm, called *HolisticTwigStack* was developed in [78]. The authors proposed a complex stack structure like *Twig2Stack* to preserve the holisticity of the twig matches, without generating intermediate solutions. However, a considerable amount of time is taken to maintain the stack structure. Dao and Gao [87] reviewed and analyzed both of the *HolisticTwigStack* and *TwigList* algorithm on processing for XML twig pattern matching. The statistics from this analysis clearly indicate that the *TwigList* algorithm seems to be significantly more efficient in the most tested cases. Recently, Li and Wang [79] proposed two novel one-phase holistic twig matching algorithms, *TwigMix* and *TwigFast*, which combine the efficient selection of useful elements introduced in *TwigStack* with the simple data lists of *TwigList* for storing final solutions. Finally, it is clear that these approaches reduce considerably the processing time. However, *Twig2Stack* reduced the intermediate results at the expense of a huge memory requirement, and it was restricted by the fan-out of the XML document.

Improvement 4: taking advantage of the properties of some labeling schemes. Most of the existing holistic twig join algorithms are based on region encoding scheme to capture the structural relationship between XML elements. However, there exist other approaches which exploit others labeling scheme. Among them, *Dewey labeling* scheme has been widely used in XML query processing. A Dewey label of a node v represents the path from the document root to the node v. Based on the Dewey labeling, Lu et al. [21] proposed *TJFast* algorithm which uses a different encoding scheme, called the extended Dewey code to combine effectively the types and identifiers of elements in a label. *TJFast* typically access much less elements than algorithms based on region encoding and can efficiently process queries with wildcards in internal nodes. More recently, based on the preliminary idea of extended Dewey labeling scheme and the *TJFast* algorithm, Lu et al. [88] proposed three novel holistic twig join algorithms *TJFastTL*, *GTJFast* and *GTJFastTL*. The first algorithm *GTJFast* allows to compactly represent the intermediate matching solutions and avoid the output of non-return nodes to reduce the I/O cost. Both *TJFastTL* and *GTJFastTL* algorithms are proposed by

extending *TJFast* and *GTJFast* algorithm based on *tag +level* streaming scheme. The authors are proved that *TJFastTL* (and *GTJFastTL*) guarantees the I/O optimality for queries with only parent–child relationships. The experimental results reported in [88] show that these algorithms are superior to existing approaches in terms of the number of scanned elements, the size of intermediate solutions and query performance.

4.1.3. *Sequence matching approaches*

As opposed to the holistic twig join algorithms, the sequence matching approaches use an indexing method to transform both XML documents and queries into sequences and evaluate queries based on sequence matching. Querying XML data is equivalent to find subsequence matches. Zezula et al. [26,76] use the *pre-order* and *post-order* ranks to linearize the tree structures and apply the sequence inclusion algorithms for strings. They proposed a novel strategy that includes three parts. Firstly, a query decomposition process is applied to transform the query twig into a set of root-to-leaf paths so that the ordered tree inclusion can be safely applied. It has to be evaluated against the data signature in the second phase. Finally, the set of answers is determined by joining compatible intermediate solutions. The experiments in [76] demonstrate the efficiency of the decomposition approach, which is especially beneficial for the large query trees, and for trees with highly selective predicates.

Two other sequence matching algorithms, *ViST* [77] and *PRIX,RM04* were proposed to avoid expensive join operations. The *ViST* method represents a major departure from previous XML indexing approaches. In fact, unlike classical index methods that decompose the query into multiple subqueries, and then join the solutions of these sub-queries to provide the final answers, *ViST* uses tree structures as the basic unit of query to avoid expensive join operations. However, the query processing in *ViST* may result in false alarms and false dismissals. These problems are explained in [90]. In this work, the authors proposed a similar subsequence matching algorithm to that of *ViST* that transforms XML documents and queries into equivalent sequences without false alarms and false dismissals. In [89], Rao and Moon proposed a new indexing XML documents and processing twig patterns in an XML database. This indexing approach transforms XML documents and twig query into sequences by *prüfer* method that constructs an one-to-one correspondence between trees and sequences. Based on this transformation, a query execution system, called *PRIX* (*Prüfer* sequences for indexing XML) is developed for indexing XML documents and processing twig queries. The matching phase is achieved by applying subsequence matching on the set of sequences in the database and performing a series of refinement phases.

4.1.4. *Other important exact XML tree algorithms*

In this section, we take a quick look over some other well-known query tree pattern processing algorithms which have been developed in recent years. Among them, some approaches have benefited from fundamental progress in node labeling schemes of XML documents (for a survey, see [91]).

In [27] for example, a twig pattern matching algorithm, called *TwigVersion* is proposed. The key idea of this approach is to compress both structural index and numbering schemes technique. *TwigVersion* is based on new version-labeling scheme that encodes all repetitive structures in XML documents. The identification of these repetitive structures matching allows to avoid a large amount of unnecessary computations. The experimental results reported in [27] show that *TwigVersion* significantly outperforms *TwigStack*, *TJFast* and *Twig2Stack* algorithms.

Recently, Izadi et al. [92] proposed a novel method, called S^3, which can selectively process the document nodes. In S^3, unlike all previous methods, path expressions are not directly executed on the XML document, but first they are evaluated against a guidance structure, called *QueryGuide*. The information extracted from the *QueryGuide* is an abstraction of the XML document. It describes the structure of XML document by its paths, such as the nodes of XML data are labeled by *Dewey labeling scheme*. The experimental results of [92] confirm that S^3 substantially outperforms *TwigStack, TJFast, and TwigList* in terms of response time, I/O overhead, and memory consumption-critical parameters.

4.1.5. *Discussion*

Much research efforts have been devoted on efficient XML query processing. As a core operation in XML data, finding all occurrences of a query pattern in XML documents attracted more and more attentions. In this section, we proposed a structured overview of the numerous recent advances in the exact tree matching for querying XML data. As mentioned before, a very large volume of algorithms were proposed in the literature, and it seems nearly impossible to consider all related works. Table 3 depicts the various approaches presented in this section. These approaches are categorized into four classes by considering three basic features: the decomposition approach, the labeling technique, and a boolean parameter which indicates if the merging phase is used in the query processing.

The structural join is the oldest method, which provides an efficient native implementation of the classical merge-join algorithm used in the relational approach. As discussed in Section 4.1.1, the main problem with the structural join based approach is that it may generate a large amount of intermediate query results because many results of individual binary relationships may not appear in the final results. Based on the containment labeling scheme, the holistic twig join algorithm partially solved the problem of larger amount intermediate solutions with decomposition-matching-merging methods. *TwigStack* is the first holistic XML twig pattern matching algorithm proposed in the literature. Specifically, the performance of *TwigStack* is better than that of *MPMGJN* and *StackTree*. The main drawback of *TwigStack* is its expensive merging phase. However, it constituted the starting point of a long series of holistic twig join approaches in order to make the twig pattern matching very efficient. From the performance study of holistic twig join family, the one-phase holistic twig matching and *TJFast* variant (and its improvements) seem to be the most competitive holistic techniques. It is however difficult to decide what is the best algorithm that performs the query processing efficiently. For example, the one-phase holistic twig join algorithm

Table 3 – Summary of XML retrieval approaches using XML exact tree matching.

Approach	Structural joins algorithms		
	Decomposition	**Merging phase**	**Labeling technique**
MPMGJN [17]	binary relationship	Yes	Region encoding
Tree-Merge [18]	binary relationship	Yes	Region encoding
Stack-Tree [18]	binary relationship	Yes	Region encoding
	Holistic twig join approach		
	Decomposition	**Merging phase**	**Labeling technique**
TwigStack [19]	root-to-leaf paths	Yes	Region encoding
TwigStackList [20]	root-to-leaf paths	Yes	Region encoding
iTwigJoin [72]	root-to-leaf paths	Yes	Region encoding
TwigBuffer [75]	root-to-leaf paths	Yes	Region encoding
TSGeneric+ [73]	root-to-leaf paths	Yes	Region encoding
TJEssential [74]	root-to-leaf paths	Yes	Region encoding
TwigStack+ [86]	root-to-leaf paths	Yes	Region encoding
Twig2Stack [22]	without decomposition	No	Region encoding
TwigList [23]	without decomposition	No	Region encoding
Holistictwigstack [78]	without decomposition	No	Region encoding
TwigMix [79]	without decomposition	No	Region encoding
TwigFast [79]	without decomposition	No	Region encoding
TJFast [21]	root-to-leaf paths	Yes	Extended Dewey labeling
TJFastTL [88]	root-to-leaf paths	Yes	Extended Dewey labeling
GTJFast [88]	root-to-leaf paths	Yes	Extended Dewey labeling
GTJFastTL [88]	root-to-leaf paths	Yes	Extended Dewey labeling
	Sequence matching approach		
	Decomposition	**Merging phase**	**Indexing technique**
PRIX [89]	Without decomposition	No	prüfer sequences
Tree signature [26,76]	root-to-leaf paths	Yes	Pre-order, Post-order
ViST [77]	without decomposition	Yes	structure-encoded sequences
Work of [90]	without decomposition	No	path labeling
	Other exact XML tree algorithms		
	Decomposition	**Merging phase**	**Labeling technique**
TwigVersion [27]	root-to-leaf paths	Yes	Dewey ID labeling
S^3 [92]	set of match patterns	Yes	Dewey ID labeling

eliminates completely the second phase of merging at the expense of huge memory requirement and it was restricted by fan-out of XML elements. Exploiting sequencing matching to speed up query evaluation is an another native idea in XML query processing. It provides significant performance enhancement over some traditional methods. However, this technique is not widely used, probably due to the limitation of XML indexing methods required to transform the tree of an XML document into a sequence.

4.2. *Approximate tree matching algorithms for XML retrieval*

There are numerous approaches for approximate tree matching in the literature of XML retrieval. Contrary to approaches presented in the previous section, the aim for these approaches is not to make a strict interpretation of structural constraints, but to select and rank elements according to their likelihood to match the queries. In case of data-centric documents, relevance is defined in terms of structural relevance while in case of text-oriented documents, it is also defined in terms of content relevance to the query. In this last case, one of the key problem of approaches doing approximate tree matching is to effectively combine results on content and on structure.

Some retrieval approaches make an *implicit* use of graph properties (one can cite diverse approaches such as [93–98]) and others *explicitly* exploit graph matching algorithms [52,99,100,85,101–103,83,81,80,84].

Approaches of the first group are more concerned with text-oriented documents, and most of them adapted traditional information retrieval approaches to structured retrieval. In this group of approaches, one can once again distinguish between three different kinds of approaches [10]:

- structural constraints can be processed by pre-generating a set of tag equivalences [93,94],
- the score of elements that match the target element[1] can be boosted [95],
- content scores can be propagated up to the document tree [96–98].

All these approaches use the tree representation of documents and queries, but do not use state-of-the art algorithms

[1] We recall that the target element in structured queries is the element to be returned.

for tree matching. For example, in the XFIRM approach [98], content and structure queries are decomposed into subqueries, composed of both a content and a structure condition, one of them indicating which type of elements should be returned (target elements). For each constraint, a score propagation is carried out, starting from leaf nodes answering the content condition and going until an element that matches the structure constraint is found. The score of result elements of each constraint is then propagated again to elements belonging to the set of targeted structures. One can find a lot of other approaches that implicitly use the tree representation of documents in the INEX proceedings [104–109].

In the following, we will focus on approaches that make an explicit use of graph matching algorithms. We classify them into two types of approaches:

- approaches based on tree inclusion,
- approaches based on the tree edit distance.

Most of these approaches do not directly use the corresponding matching algorithm: they often make some preprocessing that we detail in the following paragraph.

4.2.1. *Pre-processing of trees before the matching process*

Pre-processing of query and document trees can be done for two main reasons:

- enlarging the search space by relaxing constraints in order to return more results,
- reducing the time and space complexity of existing algorithms.

Relaxation. Enlarging the space of search is mainly done by relaxation, that can be defined as relaxing constraints. Relaxation is mainly done on queries, but can also be used on documents. When relaxing on queries, one attempts to increase the number of results by decreasing the constraints degrees [110]. Relaxation can concern both the content and structure constraints [85]. When considering structure, the graph structure can be modified by acting on edges, labels or nodes. There exist three types of structure relaxation:

1. order relaxation, which consists in reducing or deleting constraints between brother nodes, i.e., to transform ordered trees into non-ordered trees.
2. nodes relaxation, which is equivalent to rename or delete nodes in trees (in this latter case, content of deleted nodes will be attached to the father node) [110].
3. edges relaxation, which extends or adds edges between nodes. For example, a parent–child relation can be transformed into an ancestor–descendant relation.

Order relaxation is implicitly used in approximate XML retrieval, since the reading order of document is not considered as a relevance factor. Node relaxation, when deleting nodes in the tree, is often used to reduce the complexity of an approach, and we will present some approaches in the next paragraph. We detail below some approaches using edges relaxation.

Authors in [102] propose to find structural fluctuation in XML documents, i.e., different potential structures in semantically identical documents. Instead of writing verbose path expression queries that should take into account all the structural possibilities, they developed a query processing primitive called amoeba join. The aim is to match hierarchy of identical nodes but being differently nested. Fig. 9 shows examples of such hierarchies.

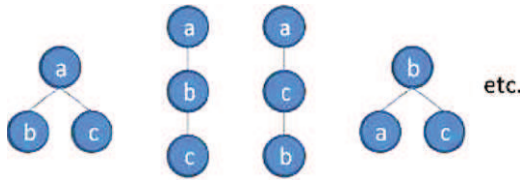

Fig. 9 – Structural fluctuation for three nodes a, b and c.

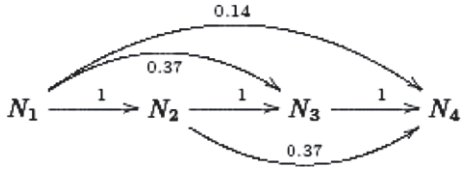

Fig. 10 – Original and final paths. Original paths are weighted with 1.

For 3 nodes a, b, c, the number of fluctuation is $3^2 = 9$, i.e., n^{n-1} if the number of nodes is n. This approach uses a query relaxation-like method for all n^{n-1} fluctuations. It can be very useful to evaluate the tree edit distance in case of structural heterogeneities.

In [99,100], authors propose to use edges relaxation to index XML documents. In their XIVIR (*XML Information retrieval based on VIRtual links*) model, they propose to represent the structure of XML documents as a bag of weighted links composed of direct and virtual links. Direct links, i.e., parent–child relations, are weighted with the maximal score of 1. New links (i.e., new edges or new paths) are weighted according to the distance d between them with the following formula:

$w = e^{(1-d(N1,N2))}$, where $d(N1, N2)$ is the distance between node $N1$ and node $N2$.

Fig. 10 shows how a path $N1 \rightarrow N2 \rightarrow N3 \rightarrow N4$ is extended with new virtual links.

Another way to obtain a represent all links in a tree is to make a transitive closure. In [103,83] for example, a fuzzy labeled graph associated with each document is constructed and weights are assigned to edges and nodes according to two approaches:

1. The first one is tag-related and attaches importance to the containment relationships that specify spatial structure of the documents,
2. The second one is structure-related and expresses the importance of XML elements and attributes.

The fuzzy labeled graph is then extended by a fuzzy transitive closure. Weights of new edges are evaluated using weights of initial edges and a t_norm operator.

Reducing the time and space complexity of existing algorithms. Reducing the time and space complexity of existing algorithms is mainly done by reducing the tree sizes of documents, using tree summaries. The main intuition behind tree

summaries is that the structure of a document can be represented in a relevant way with a smaller tree size [111,112]. Redundancy is deleted with tree summaries, but hierarchical relations can be altered. Dalamagas [111] for example proposes two rules for summarizing trees. These rules are remove nesting which is equivalent to move the subtree of a node having the same label than one of its ancestor; and remove duplicates from father–child relationship which removes the siblings having the same labels.

Tree summaries has been successfully used before applying tree edit distance in [82].

Some of the aforementioned approaches for trees preprocessing use then their own algorithm for tree matching [103,99], while others directly use state-of-the-art algorithms (tree inclusion of tree edit distance). We will present in the two next sections how those state-of-the-art algorithms have been adapted to XML retrieval. We will discuss for each of these approaches how content conditions of queries are taken into account.

4.2.2. *Approaches based on tree inclusion*

In [52] Schlieder and Naumann introduced the design and implementation of the pattern matching language ApproXQL, a query language that returns approximate answers to formal user queries. Answering queries against XML document collections is seen as an unordered tree inclusion problem. A cost model is applied to the embeddings, which helps ranking approximate matches according to their similarity to the query. Authors however did not pay attention to content conditions of queries.

The unordered tree inclusion problem is also used as referential by another approach proposed by Schlieder and Meuss [50]. Authors extend the concept of term with structural terms. Term frequency and inverse document frequency are then adapted to logical documents and structural terms. Authors argue that their "model generalizes the two main concepts on which it was based: tree matching for structured queries and the vector space model for ranking documents".

4.2.3. *Approaches based on tree edit distance*

Tree edit distance is to our knowledge the most adapted algorithm for approximate XML retrieval.

Tree edit distance is an extension of the string edit distance. The approach proposed in [80] in the SIRIUS system reduces the matching problem to a string edit distance scoring process. A modified weighted editing distance based on Levenshtein editing distance [113] on XML paths is used to approximately match the documents structure with the query structure. At the same time, for content matching, strict and fuzzy searching based on the IDF of the researched terms are used. Content and structure matching scores are then combined using a weighted linear aggregation. Experiments on the INEX 2005 test set showed the effectiveness of the proposed weighted editing distance on XML paths.

The Tai algorithm [53] for tree edit distance is used without any adaptation in [81]. Authors propose to combine a structural score and a content score to evaluate the similarity between queries and documents. The structural score is evaluated using tree edit distance. The cost of deletion of a node is considered as equal to one, where the cost of substitution is linked the semantic proximity of nodes tag names. The content score is obtained thanks to the propagation of the scores of leaf nodes elements to their ancestors. Both structural and content scores are then combined in a linear way.

As in [81], Laitang et al. [84] used a linear combination of a structural and a content score to evaluate subtrees relevance. The approach combines a classical content similarity measure with the tree edit distance. The chosen algorithm uses an optimal cover strategy to reduce the number of subgraph in memory. More precisely, the structure part is processed through a tree edit distance optimal cover strategy inspired by [114,55] works. Results on the INEX 2005 SSCAS task showed the interest of the approach.

The same authors also experimented tree summaries before applying tree edit distance in [82]. Results are comparable with those obtained with no summary, and efficiency is improved.

Tree edit distance has also been applied conjointly with relaxation. In [101,85] for example, authors propose approaches for querying a set of heterogeneous data-oriented semi-structured documents. The main problem concerns incomplete or missing data and to non-exact matching of types and structure.

To match document and query trees, they use Minimum Spanning Trees. The minimum Spanning tree [113] of a graph is the tree of minimum length connecting all its nodes, where length is the sum of the weights of the connecting edges in the tree. In the XML retrieval context, the minimal spanning tree is defined as the tree which contains query elements and a minimum number of nodes/arcs. The adaptation of the concept of Minimum Spanning Trees to structured information retrieval is composed of two steps: nodes scoring and tree matching. Nodes scoring is done in two different ways, depending on the node type. In case of inner nodes, that only have their labels as textual information, authors use the Levenshtein distance [113] between document and queries node names. In case of leaf nodes, authors use the fuzzy set theory to evaluate the membership degrees of nodes to the content conditions of queries. A structural validation of results is then done by keeping nodes that build a Minimum Spanning Tree.

The Minimum Spanning Tree approach comes from the rapprochement of relaxation and tree-edit distance (through inclusion). Even if results in terms of effectiveness are creditable, the proposed algorithm is memory consuming, especially in the second step.

4.2.4. *Discussion*

Table 4 summarizes the approaches presented in this section, regarding how they adapt formal algorithms for tree matching and how they process content conditions of queries.

Regarding the pros and cons of each approach, one can say that most of them have shown their interest in term of effectiveness ([99,80,84,82] have been evaluated in the INEX framework, see Section 5 for more details). They however do not perform well regarding efficiency (time and space complexity). A pre-processing of trees before the matching process (using tree summaries for example) can however help to reduce the matching complexity.

Table 4 – Summary of approaches using approximate tree matching.

Approach	Based on	Adaptation	Integration of content constraints
XIVIR [99]	Own matching algorithm	Query relaxation	Linear combination of content and structure scores
[83]	Own matching algorithm	Fuzzy graphs	/
[101]	Tree edit distance	Relaxation	/
ApproXQL [52]	Unordered tree inclusion	/	/
[50]	Unordered tree inclusion	/	Together with structure. Notion of structural term
SIRIUS [80]	Tree edit distance (path matching)	Relaxation	Weighted linear aggregation for content and structure matching scores
[81]	Tree edit distance	Text-oriented	Linear combination of content and structure scores
[84]	Tree edit distance	Optimal cover strategy	Linear combination of content and structure scores
[82]	Tree edit distance	Trees summaries	Linear combination of content and structure scores

5. Evaluation

Evaluation is crucial and essential to validate methods and algorithms proposed for retrieval, whether we speak of exact or approximate matching, and whether we work on text or data oriented collections. Indeed, as seen in Section 4, most of the proposed approaches for XML retrieval are adaptation of tree matching algorithms that should be carefully evaluated to see if they can be used in practice.

5.1. *Efficiency and effectiveness*

Evaluation of algorithms and approaches for tree matching can be done on two ways: one can evaluate **efficiency** and/or **effectiveness**.

Approaches for exact tree matching are directly concerned with efficiency, while those for approximate tree matching are more concerned with effectiveness.

Efficiency for exact tree matching is evaluated in terms of algorithms complexity, as well as in terms of execution time [115]. Some indications about algorithms complexity are given in Section 4. To compare systems and algorithms, some benchmarks were also created, among which we can cite [116,117] or [118]. Those benchmarks aim at evaluating XML databases on some representative tasks. In [118] more precisely, the aim is to evaluate the basic query evaluation operations, such as selections, joins and aggregations.

In this section, we will focus on the evaluation of approximate tree matching and effectiveness (i.e., the ability of systems to correctly rank relevant elements). Research on the domain has been considerably developed during the past years thanks to the INitiative for the Evaluation of XML retrieval (INEX) evaluation campaign.

5.2. *The INEX evaluation campaign*

The first INitiative for the Evaluation of XML retrieval (INEX) workshop took place in 2002. It was in 2012 the eleventh year of this now established evaluation forum for XML Information Retrieval. During these 11 editions, about 100 different worldwide organizations participated to the proposed tracks [119].

The INEX campaign provides to participants an infrastructure to evaluate XML IR approaches:

- search tasks,
- large structured test collections (documents and queries),
- and scoring methods.

Participants, besides running queries on their systems, also provide test queries and relevance judgments to the evaluation infrastructure: this is why INEX is often seen as a collaborative effort.

Among the proposed search tracks over the 11 campaigns, one can cite the Ad Hoc, Multimedia, Book, XML-Mining, Relevance Feedback, Efficiency or Tweet Contextualization tracks.

Among those tracks, the ad hoc, data-centric and efficiency ones can be used to evaluate approximate tree matching algorithms. They are detailed in the following paragraphs.

5.2.1. *The ad hoc track*

Search tasks and queries

The ad hoc track aims at evaluating how a digital library is typically used: information is in a fixed collection of XML documents and is retrieved using a set of topics.

Two main collections composed of text-oriented documents were used for the ad hoc track:

- the IEEE collection, composed of about 180,000 scientific articles with extensive XML-markup from 21 IEEE Computer Society journals published in 1995–2002,
- the Wikipedia collection, composed of Wikipedia articles. The original collection contained about 660,000 articles [120], and was dumped again in 2009 with annotation from YAGO [121]. It now contains about 2660,000 articles.

In previous years, a distinction was made between Content Only (CO) and Content And Structure (CAS) topics. This led to two main search sub-tasks: the CO one and the CAS one.

Since approximate tree matching considers queries as trees, only the CAS one can be used to evaluate such approaches. Over the years, many search tasks were associated to CAS queries. In 2005 for example, 4 sub-tasks were run:

- VVCAS: The target element and support element constraints are considered as vague.
- SVCAS: The target element constraint is considered as strict, but the support element constraints are considered as vague.
- VSCAS: The target element constraint is considered as vague, but the support element constraints should be followed strictly.
- SSCAS: Both the target element constraint and the support element constraint are considered as strict.

From 2006, CAS queries disappeared and were merged with CO ones in what is called in the INEX terminology CO+S queries. The aim was to have only one topic type for all search tasks.

Structure-oriented approaches can still be evaluated with the ad hoc track, but since 2006, structure interest is no more a research priority for the ad hoc track organizers. To overcome this limitation, the data-centric track appears in 2010.

Evaluation metrics

Metrics used over the years to evaluate the ad hoc track evolved with the definition of the task. Our aim here is not to list all the metrics used during the 9 years of the task (the ad hoc track last ran in 2010), but to give some pointers on metrics used to evaluate XML retrieval approaches.

Metrics evaluating XML retrieval should mainly deal with the following two problems:

- the retrieval unit is not a whole document, but a document part (i.e., an element of the XML tree);
- systems can return multiple nested elements that contain the same information (this is also know as the *overlap problem* [122]).

As returned elements can be of different granularities, relevance in XML retrieval is defined according to two dimensions:

- Exhaustivity (*e*): an element is very exhaustive if it contains all the information required by the query,
- Specificity (*s*): an element is very specific if all its content concerns the query.

These two dimensions of relevance must then be quantized (via quantization functions) into a single relevance value that represents the level of relevance of an element. We give below two examples of quantization functions:

- the strict quantization evaluates if a given retrieval approach is able of retrieving highly exhaustive and highly specific document components,

$$f_{strict}(e, s) = \begin{cases} 1 & \text{if } e = 2 \text{ and } s = 1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

- the generalized quantization evaluates document components according to their degree of relevance.

$$f_{generalised}(e, s) = e * s. \quad (2)$$

In 2005 for example, $e = 2$ and $s = 1$ were respectively the highest degrees of specificity and exhaustivity ($e \in \{0, 1, 2\}$ and $s \in [0, 1]$). Since 2006, all relevant elements were considered as highly exhaustive, and specificity was computed as the ratio of the number of relevant characters contained within the XML element to the total number of characters contained by the element.

The main metrics used in INEX are based on the extended cumulated gain (XCG) [123]. The aim of these metrics is to take into account the dependency of XML elements, that is to take into account overlap (nested elements) and near misses (returned elements that are near relevant ones). Two metrics are included in the XCG metrics: the user-oriented measures of normalized extended accumulated gain (nXCG) which accumulates the relevance score of retrieved elements along the ranked list, and the system-oriented effort-precision/gain-recall measures (ep/gr).

Other metrics used in the context of INEX can be found in [124] (iP metrics), [125] (T2I metric), [126] (EPRUm metric) or HiXEval [127] (HiXEval). A study on these metrics is also available in [128].

5.2.2. *The data-centric track*

The data centric track was introduced at INEX in 2010 (and last ran in 2011). This task aims at studying if whole document retrieval is effective in case of highly structured document collections.[2]

The used collection is a snapshot of the IMDB taken early in 2010.[3] Each XML file contains information about one movie or person. In total, more than 4400,000 XML files were generated, representing among others movies and actors [129].

In its first year, the track focused on ad hoc retrieval from XML data. The task, as defined in [129] "was to return a ranked list of results estimated relevant to the user's information need. The content of the collections of nodes was not permitted to overlap. This is similar to the focused task in the ad hoc track, but using a data-centric XML collection and allowing the construction of a result (i.e., a collection of nodes) from different parts of a single document or even multiple documents".

28 highly structured queries were proposed by participants. Evaluation were done using the MAiP [124] and MAgP T2I [125] metrics (thus without measuring aggregation since this is still not clear how to measure it).

Concerning 2010 results, one can say the following about the proposed approaches:

- structure was directly considered by one participant [130] (i.e., they directly used structured queries), but results were disappointing.
- some other approaches experimented to directly generate structured queries from keyword-based queries without trying to use the proposed CAS queries and in order to query an XML DBMS [131,132].

In 2011, the task evolved and proposed two sub-tasks [133]. The ad hoc subtask was more concerned with the retrieval of documents (instead of elements), even if structure was strongly taken into account. A new subtask, called faceted search, proposed to participants to help the user navigating through results by giving him/her a list of facet values representing the result documents.

5.2.3. *The efficiency track*

The aim of the efficiency track (launched in 2008 and 2009) was to evaluate the effectiveness and efficiency of XML ranked retrieval approaches on real data and real queries [134].

Participants had to run their system with among others queries with a deeply nested structure on the Wikipedia

[2] It was indeed shown during the past editions of INEX that document retrieval helps element retrieval. This conclusion is maybe due to the collection used for test (scientific articles or Wikipedia pages).

[3] www.imdb.com.

collection [121]. Efficiency was evaluated with runtime, CPU time and I/O statistics to secondary memory, taking into account general system statistics (#CPUs, memory, disk configuration, caching options, distribution). Effectiveness was evaluated with more traditional metrics (iP, MAiP, etc. . . .).

The best systems achieved interactive retrieval times for ad hoc search, with a result quality comparable to the best runs in the ad hoc track [134].

6. Discussions and future research directions

We proposed in this paper a comprehensive survey about tree matching for XML information retrieval. The matching problem was addressed from both an exact and an approximate point of view.

In the exact matching part of the paper, we reviewed structural join methods, string matching methods and other diverse methods. The structural join approach is mainly used for processing XML twig queries. Structural Join as the oldest method decomposes a query twig pattern into its binary relationships (Parent–Child relationship, Ancestor–Descendant relationship) and executes them separately. A major drawback of this approach is that many intermediate results may not be part of any final answer. Of course, holistic twig join algorithms are good candidates for physical operators supporting query evaluation in XDBMSs. However, they only provide for a small fraction of the functionality required by complete XPath and XQuery processors (e.g., no processing of axes other than child and descendant; no processing of order-based queries). Therefore, the development of new structural join algorithms is still valuable, because they can act as complementary operators in case the restricted functionality of twig joins is too small, or as alternatives if they promise faster query evaluation. It would also be interesting to study the parallelism of holistic algorithms on a multi-core system in order to maximize the computing performance of query processing in a large XML database.

Concerning the approximate matching part of the paper, approaches for XML retrieval presented here are mainly based on the tree edit distance. They obtain relatively good results in terms of effectiveness, but issues concerning memory and efficiency need still to be solve. The relatively small number of approaches presented can be explained by two reasons. In an information retrieval point of view:

- it is more simple to adapt effective and efficient IR algorithms to structure than the alternative, i.e., it is easier to start with content conditions and filter results on structural ones than starting from structural constraints,
- Many discussions were conducted between IR researchers to state the usefulness of structural queries on text-oriented documents (whether structured queries are real user needs and whether structural constraints help retrieval). No strong conclusion can be drawn on this usefulness [135–137]. This may explain the actual lack of interest of the IR community on the subject: a very small number of papers on XML retrieval have been published during the last two years in the main Information Retrieval conferences (SIGR, CIKM and ECIR).

XML tree matching algorithms described in this paper were presented in an XML retrieval perspective. Applications that also use XML tree-matching are however numerous:

- version control, change management and date warehousing,
- semi-structured data integration,
- classification/clustering of XML documents gathered from the web against a set of XML grammars declared in an XML database.

One will find in [6] a complete survey on those applications. At last, to conclude this paper we highlight two important research directions in the graph theory community that may lead to new XML tree matching algorithms:

- Graph *embedding/packing* are a well known field in graph theory. Many conjectures and results have been obtained to solve the graph embedding/packing problem for unlabeled graphs [138]. The graph matching problem can be naturally stated as packing of labeled graph. If we consider the example of XML documents that are modeled as ordered labeled tree, we can see that the querying of an XML document is equivalent to the search of an embedding of the query in the data tree. Therefore, it would be interesting to exploit existing tree embedding techniques in order to define a coherent similarity measure between two labeled trees.
- Another line of research that might be investigated is quantum techniques for tree matching. In the literature, quantum algorithms have attracted considerable attention in the theoretical computer science community because of the considerable speed up over classical algorithms they achieve [139]. For a practical perspective of tree matching problem, several aspects remain to be investigated. Indeed, quantum versions of matching approaches based on decomposition, combinatorial of words can be developed. Nevertheless, the formalization of such solutions may be very hard choice, but very promising.

REFERENCES

[1] W3C, EXtensible Markup Language (XML) 1.0, Tech. Rep., World Wide Web Consortium (W3C), Recommendation, February 1998.
[2] W3C XML Web page. http://www.w3.org/XML/.
[3] M. Lalmas, R. Baeza-Yates, In: R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley Professional, 2 edition (February 10, 2011), Ch. Structured Text Retrieval.
[4] W3C, DOM level 1 (Document Object Model), Tech. Rep., World Wide Web Consortium (W3C), Recommendation, October 1998.
[5] M. Lalmas, XML Retrieval, Synthesis Lectures on Information Concepts, Retrieval, and Services, Morgan & Claypool Publishers, 2009.
[6] J. Tekli, R. Chbeir, K. Yétongnon, An overview on XML similarity: background, current trends and future directions, Comput. Sci. Rev. 3 (3) (2009) 151–173.
[7] L. Liu, M.T. Özsu (Eds.), Encyclopedia of Database Systems, Springer, US, 2009.
[8] M. Lalmas, R. Baeza-Yates, Structured document retrieval, in: Encyclopedia of Database Systems, Springer, 2009, pp. 2867–2868.

[9] G. Kazai, N. Govert, M. Lalmas, N. Fuhr, The INEX evaluation initiative, in: Intelligent Search on XML data, Applications, Languages, Models, Implementations and Benchmarks, Springer, 2003, pp. 279–293.
[10] A. Trotman, Processing structural constraints, in: Encyclopedia of Database Systems, Springer, 2009, pp. 2191–2195.
[11] S. Amer-Yahia, M. Lalmas, XML search: languages, INEX and scoring, ACM SIGMOD Rec. 35 (4) (2006) 16–23.
[12] W3C, XML Path Language (XPath) 2.0, Tech. Rep., World Wide Web Consortium (W3C), Recommendation, January 2007.
[13] A. Trotman, B. Sigurbjornsson, Narrowed EXtended XPath I (NEXI), in: Proceedings of the 3rd Workshop of the Initiative for the Evaluation of XML retrieval (INEX), 2004, pp. 16–40.
[14] A. Campi, E. Damiani, S. Guinea, S. Marrara, G. Pasi, P. Spoletini, A fuzzy extension of the XPath query language, Journal of Intelligent Information Systems 33 (3) (2009) 285–305.
[15] W3C, XQuery 1.0 : An XML query language, Tech. Rep., World Wide Web Consortium (W3C), Recommendation, January 2007.
[16] W3C, XQuery 1.0 and XPath 2.0 Full-Text 1.0, Tech. Rep., World Wide Web Consortium (W3C), Note, January 2011.
[17] C. Zhang, J. Naughton, D. Dewitt, Q. Luo, G. Lohman, On supporting containment queries in relational database management systems, in: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, California, USA, 2001, pp. 425–426.
[18] S. Al-Khalifa, H. Jagadish, N. Koudas, J. Patel, D. Srivastava, Y. Wu, Structural joins: a primitive for efficient XML query pattern matching, in: Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, 2002, pp. 141–152.
[19] N. Bruno, N. Koudas, D. Srivastava, Holistic twig joins: optimal XML pattern matching, in: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, 2002, pp. 310–321.
[20] J. Lu, T. Chen, T.W. Ling, Efficient processing of XML twig patterns with parent child edges: a look-ahead approach, in: Proceedings of the 13th ACM International Conference on Information and Knowledge Management, CIKM, Washington D.C., USA, 2004, pp. 533–542.
[21] J. Lu, T.W. Ling, C.Y. Chan, T. Chen, From region encoding to extended Dewey: on efficient processing of XML twig pattern matching, in: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB, Trondheim, Norway, 2005, pp. 193–204.
[22] S. Chen, H.G. Li, J. Tatemura, W.P. Hsiung, D. Agrawal, K.S. Candan, Twig2Stack: bottom-up processing of generalized-tree-pattern queries over XML documents, in: Proceedings of the 32nd International Conference on Very Large Data Bases, in: VLDB'06, 2006, pp. 283–294.
[23] L. Qin, J.X. Yu, B. Ding, Twiglist: make twig pattern matching fast, in: Proceedings of the 12th International Conference on Database Systems for Advanced Applications, DASFAA, DASFAA'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 850–862.
[24] S.C. Haw, C.S. Lee, TwigX-Guide: an efficient twig pattern matching system extending dataguide indexing and region encoding labeling, J. Inf. Sci. Eng. 25 (2) (2009) 603–617.
[25] D. Shasha, J.T.-L. Wang, H. Shan, K. Zhang, Atreegrep: approximate searching in unordered trees, in: Proceedings of the 14th International Conference on Scientific and Statistical Database Management, SSDBM, Edinburgh, Scotland, UK, 2002, pp. 89–98.
[26] P. Zezula, F. Mandreoli, R. Martoglia, Tree signatures and unordered XML pattern matching, in: SOFSEM: Theory and Practice of Computer Science, 30th Conference on Current Trends in Theory and Practice of Computer Science, Merin, Czech Republic, 2004, pp. 122–139.
[27] X. Wu, G. Liu, XML twig pattern matching using version tree, Data Knowledge Eng. 64 (2008) 580–599.
[28] J. Yao, M.Z. II, A fast tree pattern matching algorithm for XML query, in: 2004 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2004, Beijing, China, 2004, pp. 235–241.
[29] P. Ogilvie, Retrieval using structure for question answering, in: Proceedings of the First Twente Data Management Workshop; XML databases and Information retrieval, 2004, pp. 15–23.
[30] A. Trotman, B. Sigurbjornsson, NEXI, now and next, in: Proceedings of the 3rd Workshop of the Initiative for the Evaluation of XML retrieval, INEX, 2004, pp. 41–53.
[31] P. Buneman, S. Davidson, G. Hillebrand, D. Suciu, A query language and optimization techniques for unstructured data, in: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, SIGMOD'96, ACM, New York, NY, USA, 1996, pp. 505–516.
[32] A. Levy, M. Fernandez, D. Suciu, D. Florescu, A. Deutsch, XMLQL: a query language for XML, Tech. Rep., World Wide Web Consortium technical report, Number NOTE- xml-ql-19980819, 1998.
[33] J. Robie, J. Lapp, D. Schach, XML query language (XQL), in: Proceedings of W3C QL98 (Query Languages 98), Massachussets, USA, 1998.
[34] D. Chamberlin, J. Robie, D. Florescu, Quilt: an XML query language for heterogeneous data sources, in: The World Wide Web and Databases, in: Lecture Notes in Computer Science, vol. 1997, 2001, pp. 1–25.
[35] D. Chamberlin, P. Fankhauser, et al. XML Query Use Cases, Tech. Rep., World Wide Web Consortium (W3C) Group Note, March 2007.
[36] C. Botev, J. Shanmugasundaram, XQuery full-text, in: Encyclopedia of Database Systems, Springer, 2009, pp. 3665–3671.
[37] W3C, XQuery 1.0 and XPath 2.0 Full-Text 1.0 Use Cases, Tech. Rep., World Wide Web Consortium (W3C) Group Note, January 2011.
[38] R. van Zwol, J. Baas, H. van Oostendorp, F. Wiering, Bricks: the building blocks to tackle query formulation in structured document retrieval, in: 28th European Conference on IR Research, ECIR 2006, London, UK, 2006, pp. 314–325.
[39] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, L. Tanca, XML-GL: a graphical language for querying and restructuring XML documents, in: Proceedings of the Eighth International Conference on World Wide Web, WWW'99, Elsevier North-Holland, Inc., New York, NY, USA, 1999, pp. 1171–1187.
[40] C.M. Hoffmann, M.J. O'Donnell, Pattern matching in trees, J. ACM (1982) 68–95.
[41] A.V. Aho, M.J. Corasick, Efficient string matching: an aid to bibliographic search, Commun. ACM (1975) 333–340.
[42] D.R. Chase, An improvement to bottom-up tree pattern matching, in: POPL'87, 1987, pp. 168–177.
[43] J. Cai, R. Paige, R.E. Tarjan, More efficient bottom-up tree pattern matching, in: CAAP'90, 1990, pp. 72–86.
[44] J. Burghardt, A tree pattern matching algorithm with reasonable space requirements, in: CAAP'88, 1988, pp. 1–15.
[45] S.R. Kosaraju, Efficient tree pattern matching, in: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, USA, 1989, pp. 178–183.
[46] M. Dubiner, Z. Galil, E. Magen, Faster tree pattern matching, J. ACM 41 (1994) 205–213.
[47] R. Cole, R. Hariharan, Tree pattern matching and subset matching in randomized $o(n \log^3 m)$ time, in: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, 1997, pp. 66–75.

[48] R. Cole, R. Hariharan, Tree pattern matching to subset matching in linear time, SIAM J. on Computing 32 (2003) 1056–1066.
[49] P. Bille, A survey on tree edit distance and related problems, Theoret. Comput. Sci. (2005) 217–239.
[50] T. Schlieder, H. Meuss, Querying and ranking XML documents, JASIST 53 (6) (2002) 489–503.
[51] L.H. Yang, M.L. Lee, W. Hsu, Finding hot query patterns over an XQuery stream, VLDB J (2004) 318–332.
[52] T. Schlieder, Approximate tree embedding for querying XML data, Proceedings of ACM SIGIR workshop on XML and information retrieval, Athens, Greece, 2000.
[53] K.C. Tai, The tree-to-tree correction problem, J. ACM 26 (1979) 422–433.
[54] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM J. Comput (1989) 1245–1262.
[55] P.N. Klein, Computing the edit-distance between unrooted ordered trees, in: Proceedings of the 6th Annual European Symposium on Algorithms, in: ESA'98, Springer-Verlag, London, UK, 1998, pp. 91–102.
[56] W. Chen, New algorithm for ordered tree-to-tree correction problem, J. Algorithms 40 (2001) 135–158.
[57] S. Dulucq, H. Touzet, Decomposition algorithms for the tree edit distance problem, J. Discrete Algorithms (2005) 448–471.
[58] E.D. Demaine, S. Mozes, B. Rossman, O. Weimann, An optimal decomposition algorithm for tree edit distance, in: ICALP'07, 2007, pp. 146–157.
[59] D.E. Knuth, The Art of Computer Programming, Volume i: Fundamental Algorithms, Third ed., Addison-Wesley, 1997.
[60] P. Kilpelainen, H. Mannila, Ordered and unordered tree inclusion, SIAM J. Comput. (1995) 340–356.
[61] T. Richter, A new algorithm for the ordered tree inclusion problem, in: Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching, CPM'97, 1997, pp. 150–166.
[62] W. Chen, More efficient algorithm for ordered tree inclusion, J. Algorithms 26 (1998) 370–385.
[63] P. Bille, I.L. Gortz, The tree inclusion problem: in optimal space and faster, in: 32nd International Colloquium on Automata, Languages and Programming, ICALP, 2005, pp. 66–77.
[64] Y. Chen, Y. Chen, A new tree inclusion algorithm, Inf. Process. Lett. 98 (2006) 253–262.
[65] H.L. Cheng, B.F. Wang, On Chen and Chen's new tree inclusion algorithm, Inf. Process. Lett. (2007) 14–18.
[66] Y. Chen, Y. Chen, A new top-down algorithm for tree inclusion, in: Proceedings of the 2010 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2010, pp. 293–300.
[67] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. ACM 21 (1974) 168–173.
[68] T. Jiang, L. Wang, K. Zhang, Alignment of trees—an alternative to tree edit, Theoret. Comput. Sci. (1995) 137–148.
[69] T. Kuboyama, K. Shin, T. Miyahara, H. Yasuda, A theoretical analysis of alignment and edit problems for trees, in: ICTCS'05, 2005, pp. 323–337.
[70] J. Jansson, A. Lingas, A fast algorithm for optimal alignment between similar ordered trees, in: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching, CPM'01, 2001, pp. 232–240.
[71] L. Wang, J. Zhao, Parametric alignment of ordered trees, Bioinformatics (2003) 2237–2245.
[72] T. Chen, J. Lu, T.W. Ling, On boosting holism in XML twig pattern matching using structural indexing techniques, in: SIGMOD 05, 2005, pp. 455–466.
[73] H. Jiang, W. Wang, H. Lu, J.X. Yu, Holistic twig joins on indexed XML documents, in: Proceedings of VLDB, 2003, pp. 273–284.
[74] G. Li, J.F.Y. Zhang, L. Zhou, Efficient holistic twig joins in leaf-to-root combining with root-to-leaf way, DASFAA, Bangkok, Thailand, 2007, pp. 834–849.
[75] J. Li, J. Wang, TwigBuffer: avoiding useless intermediate solutions completely in twig joins, in: Proceedings of the 13th International Conference on Database Systems for Advanced Applications, in: DASFAA'08, 2008, pp. 554–561.
[76] P. Zezula, G. Amato, F. Debole, F. Rabitti, Tree signatures for XML querying and navigation, in: Xsym'03, 2003, pp. 149–163.
[77] H. Wang, S. Park, W. Fan, P.S. Yu, Vist: a dynamic index method for querying XML data by tree structures, in: SIGMOD Conference'03, 2003, pp. 110–121.
[78] Z. Jiang, C. Luo, W.C. Hou, Q. Zhu, D. Che, Efficient processing of XML twig pattern: a novel one-phase holistic solution, in: DEXA 07, 2007, pp. 87–97.
[79] J. Li, J. Wang, Fast matching of twig patterns, in: Proceedings of the 19th International Conference on Database and Expert Systems Applications, in: DEXA'08, 2008, pp. 523–536.
[80] E. Popovici, G. Ménier, P.-F. Marteau, SIRIUS: a lightweight XML indexing and approximate search system at INEX 2005, in: INEX, 2005, pp. 321–335.
[81] M.D. Le, K. Pinel-Sauvagnat, Utilisation de la distance d'édition pour l'appariement sémantique de documents XML, in: Atelier GAOC, Conférence EGC 2010, Tunisia, 2010.
[82] C. Laitang, M. Boughanem, K. Pinel-Sauvagnat, XML information retrieval through tree edit distance and structural summaries, in: Asia Information Retrieval Society Conference, AIRS, Springer, Dubai, United Atab Emirates, 2011, pp. 73–83.
[83] E. Damiani, B. Oliboni, L. Tanca, Fuzzy techniques for XML data smushing, in: B. Reusch (Ed.), Fuzzy Days, in: Lecture Notes in Computer Science, vol. 2206, Springer, 2001, pp. 637–652.
[84] C. Laitang, K. Pinel-Sauvagnat, M. Boughanem, Utilisation de la théorie des graphes et de la distance d'édition pour la recherche d'information sur documents XML, in: Proceedings of the 8th Annual Conférence en Recherche d'Information et Applications, CORIA, Avignon, France, 2011.
[85] A. Alilaouar, Interrogation flexible de documents semi-structurés, Ph.D. thesis, Université Paul Sabatier, Toulouse, France, October 2007.
[86] J. Zhou, M. Xie, X. Meng, Twigstack+: holistic twig join pruning using extended solution extension, Wuhan Univ. J. Natural Sci. 12 (5) (2007) 855–860.
[87] D.B. Dao, J. Cao, A glance on current XML twig pattern matching algorithms, in: ICCSA (2)'08, 2008, pp. 307–321.
[88] J. Lu, X. Meng, T.W. Ling, Indexing and querying XML using extended dewey labeling scheme, Data Knowl. Eng. 70 (2011) 35–59.
[89] P. Rao, B. Moon, Prix: indexing and querying XML using prufer sequences, in: ICDE'04, 2004, pp. 288–300.
[90] H. Wang, X. Meng, On the sequencing of tree structures for XML indexing, in: Proceeding of ICDE, 2005, pp. 372–383.
[91] S.C. Haw, C.S. Lee, Node labeling schemes in XML query optimization: a survey and trends, IETE Tech. Rev. 26 (2) (2009) 88–100.
[92] S.K. Izadi, T. Harder, M.S. Haghjoo, S^3: evaluation of tree-pattern XML queries supported by structural summaries, in: Proceedings of Data and Knowledge Engineering, 2009, pp. 126–145.
[93] Y. Mass, M. Mandelbrod, Using the INEX environment as a test bed for various user models for XML retrieval, in: INEX, 2005, pp. 187–195.
[94] V. Mihajlovic, G. Ramirez, T. Westerveld, D. Hiemstra, H. Blok, A.P. de Vries, TIJAH scratches INEX 2005: vague element selection, image search, overlap, and relevance feedback, in: INEX, 2005, pp. 72–87.

[95] R. van Zwol, B3-sdr and effective use of structural hints, in: INEX, 2005, pp. 146–160.

[96] M. Theobald, R. Schenkel, G. Weikum, Topx and XXL at INEX 2005, in: INEX, 2005, pp. 282–295.

[97] G. Hubert, XML retrieval based on direct contribution of query components, in: INEX, 2005, pp. 172–186.

[98] K. Pinel-Sauvagnat, M. Boughanem, C. Chrisment, Answering content-and-structure-based queries on XML documents using relevance propagation, in: Information Systems, Elsevier, 2006, pp. 172–186. Special Issue SPIRE 2004 31.

[99] M.B. Aouicha, M. Tmar, M. Boughanem, M. Abid, XML information retrieval based on tree matching, in: IEEE International Conference on Engineering of Computer Based Systems, ECBS, Belfast, Ireland, 2008, pp. 499–500.

[100] M.B. Aouicha, Une approche algébrique pour la recherche d'information structurée, Ph.D. thesis, Université Paul Sabatier, Toulouse, France, 2009.

[101] A. Alilaouar, F. Sédes, Fuzzy querying of XML documents, in: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM, Compiègne, France, 2005, pp. 11–14.

[102] T.L. Saito, S. Morishita, Amoeba join: overcoming structural fluctuations in XML data, in: Ninth International Workshop on the Web and Databases, WebDB 2006, Chicago, Illinois, USA, 2006, pp. 38–43.

[103] E. Damiani, L. Tanca, F.A. Fontana, Fuzzy XML queries via context-based choice of aggregations, Kybernetika 36 (6) (2000) 635–655.

[104] N. Fuhr, M. Lalmas, S. Malik, G. Kazai (Eds.), Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl Castle, Germany, in: Lecture Notes in Computer Science, vol. 3977, Springer, 2006.

[105] N. Fuhr, M. Lalmas, A. Trotman (Eds.), Comparative evaluation of XML information retrieval systems, in: 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006, Dagstuhl Castle, Germany, in: Lecture Notes in Computer Science, vol. 4518, Springer, 2007.

[106] N. Fuhr, J. Kamps, M. Lalmas, A. Trotman (Eds.), Focused access to XML documents, in: 6th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2007, Dagstuhl Castle, Germany, in: Lecture Notes in Computer Science, vol. 4862, Springer, 2008.

[107] S. Geva, J. Kamps, A. Trotman (Eds.), Advances in focused retrieval, in: 7th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2008, Dagstuhl Castle, Germany, in: Lecture Notes in Computer Science, vol. 5631, Springer, 2009.

[108] S. Geva, J. Kamps, A. Trotman (Eds.), Focused retrieval and evaluation, in: 8th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2009, Brisbane, Australia, in: Lecture Notes in Computer Science, vol. 6203, Springer, 2010.

[109] S. Geva, J. Kamps, R. Schenkel, A. Trotman (Eds.), Comparative evaluation of focused retrieval, in: 9th International Workshop of the Inititative for the Evaluation of XML Retrieval, INEX 2010, Vugh, The Netherlands, December 13–15, 2010, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 6932, Springer, 2011.

[110] S. Amer-Yahia, S. Cho, D. Srivastava, Tree pattern relaxation, in: EDBT02, Prague, Czech Republic, 2002, pp. 496–513.

[111] T. Dalamagas, T. Cheng, K.J. Winkel, T.K. Sellis, Clustering XML documents using structural summaries, in: EDBT Workshops, 2004, pp. 547–556.

[112] T. Dalamagas, T. Cheng, K.J. Winkel, T.K. Sellis, A methodology for clustering XML documents by structure, Inf. Syst. 31 (3) (2006) 187–228.

[113] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Phys. Dokl. 10 (8) (1966) 707–710.

[114] S. Dulucq, H. Touzet, Analysis of tree edit distance algorithms, in: Proceedings of Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, 2003, pp. 83–95.

[115] S. Al-Khalifa, XML query evaluation, Ph.D. thesis, University of Michigan, Ann Arbor, USA, 2005.

[116] A. Schmidt, F. Waas, M.L. Kersten, M.J. Carey, I. Manolescu, R. Busse, XMark: a benchmark for XML data management, in: Proceedings of 28th International Conference on Very Large Data Bases, VLDB, Hong Kong, China, Morgan Kaufmann, 2002, pp. 974–985.

[117] B.B. Yao, M.T. Özsu, N. Khandelwal, XBench benchmark and performance testing of XML DBMSs, in: Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, IEEE Computer Society, Boston, MA, USA, 2004, pp. 621–633.

[118] K. Runapongsa, J.M. Patel, H.V. Jagadish, Y. Chen, S. Al-Khalifa, The Michigan benchmark: towards XML query performance diagnostics, Inf. Syst. 31 (2) (2006) 73–97.

[119] R.S. Shlomo Geva, Jaap Kamps, Inex'12 workshop pre-proceedings, 2012.

[120] L. Denoyer, P. Gallinari, The wikipedia XML corpus, SIGIR Forum 40 (1) (2006) 64–69.

[121] R. Schenkel, F. Suchanek, G. Kasneci, YAWN: a semantically annotated Wikipedia XML corpus, in: 12. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web, BTW 2007, Vol. 103, 2007, pp. 277–291.

[122] G. Kazai, M. Lalmas, A.P. de Vries, The overlap problem in content-oriented XML retrieval evaluation, in: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2004, Sheffield, UK, 2004, pp. 72–79.

[123] G. Kazai, M. Lalmas, eXtended cumulated gain measures for the evaluation of content-oriented XML retrieval, ACM Trans. Inf. Syst. 24 (4) (2006) 503–542.

[124] J. Kamps, J. Pehcevski, G. Kazai, M. Lalmas, S. Robertson, INEX 2007 evaluation measures, in: INEX, 2007, pp. 24–33.

[125] A.P. de Vries, G. Kazai, M. Lalmas, Tolerance to irrelevance: a user-effort evaluation of retrieval systems without predefined retrieval unit, in: Proceedings of RIAO 2004, Avignon, France, 2004, pp. 463–473.

[126] B. Piwowarski, P. Gallinari, G. Dupret, Precision recall with user modeling (prum): application to structured information retrieval, ACM Trans. Inf. Syst. 25 (1) (2007).

[127] J. Pehcevski, J.A. Thom, Hixeval: highlighting XML retrieval evaluation, in: INEX, 2005, pp. 43–57.

[128] J. Pehcevski, B. Piwowarski, Evaluation metrics for structured text retrieval, in: Encyclopedia of Database Systems, 2009, pp. 1015–1024.

[129] A. Trotman, Q. Wang, Overview of the INEX 2010 data centric track, in: INEX 2010 pre-proceedings, 2010, pp. 128–137.

[130] D.V. Ayala, D. Pinto, C. Balderas, M. Tovar, BUAP: a first approach to the data-centric track of INEX 2010, in: INEX 2010 pre-proceedings, 2010, pp. 159–168.

[131] F. Hummel, A. da Silva, M. Moro, A. Laender, Automatically generating structured queries in XML keyword search, in: INEX 2010 pre-proceedings, 2010, pp. 138–149.

[132] Q. Li, Q. Wang, S. Wang, Inferring query pattern for XML keyword retrieval, in: INEX 2010 pre-proceedings, 2010, pp. 150–158.

[133] Q. Wang, G. Ramírez, M.M. Marx, M. Theobald, J. Kamps, Overview of the inex 2011 data-centric track, in: Focused Retrieval and Evaluation: 10th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2011, in: Lecture Notes in Computer Science, vol. 7424, Springer, Hofgut Imsbach, Theley, Germany, 2012.
[134] R. Schenkel, M. Theobald, Overview of the INEX 2009 Efficiency Track, in: INEX, 2009, pp. 200–212.
[135] A. Trotman, M. Lalmas, Why structural hints in queries do not help XML-retrieval, in: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2006, Seattle, Washington, USA, 2006, pp. 711–712.
[136] K. Sauvagnat, M. Boughanem, C. Chrisment, Why using structural hints in XML retrieval? in: Flexible Query Answering Systems, 7th International Conference, FQAS 2006, Milan, Italy, 2006, pp. 197–209.
[137] C. Laitang, K. Pinel-Sauvagnat, M. Boughanem, DTD based costs for TreeEdit distance in Structured Information Retrieval, in: European Conference on Information Retrieval (ECIR), Moscou, Russie, 25–27 March 2013.
[138] S. Stahl, The embeddings of a graph-a survey, J. Graph Theory 2 (1978) 275–298.
[139] C. Dürr, M. Heiligman, P. Hoyer, M. Mhalla, Quantum query complexity of some graph problems, in: Proceedings of ICALP'04, 2004, pp. 481–493.