A Survey of XML Tree Patterns

Marouane Hachicha and Jérôme Darmont, *Member, IEEE Computer Society*

Abstract—With XML becoming an ubiquitous language for data interoperability purposes in various domains, efficiently querying XML data is a critical issue. This has lead to the design of algebraic frameworks based on tree-shaped patterns akin to the tree-structured data model of XML. Tree patterns are graphic representations of queries over data trees. They are actually matched against an input data tree to answer a query. Since the turn of the twenty-first century, an astounding research effort has been focusing on tree pattern models and matching optimization (a primordial issue). This paper is a comprehensive survey of these topics, in which we outline and compare the various features of tree patterns. We also review and discuss the two main families of approaches for optimizing tree pattern matching, namely pattern tree minimization and holistic matching. We finally present actual tree pattern-based developments, to provide a global overview of this significant research topic.

Index Terms—XML Querying, Data Tree, Tree Pattern, Tree Pattern Query, Twig Pattern, Matching, Containment, Tree Pattern Minimization, Holistic Matching, Tree Pattern Mining, Tree Pattern Rewriting.

1 Introduction

Since its inception in 1998, the eXtended Markup Language (XML) [1] has emerged as a standard for data representation and exchange over the Internet, as many (mostly scientific, but not only) communities adopted it for various purposes, e.g., mathematics with MathML [2], chemistry with CML [3], geography with GML [4] and e-learning with SCORM [5], just to name a few. As XML became ubiquitous, efficiently querying XML documents quickly appeared primordial and standard XML query languages were developed, namely XPath [6] and XQuery [7]. Research initiatives also complemented these standards, to help fulfill user needs for XML interrogation, e.g., XML algebras such as TAX [8] and XML information retrieval [9].

Efficiently evaluating path expressions in a tree-structured data model such as XML's is crucial for the overall performance of any query engine [10]. Initial efforts that mapped XML documents into relational databases queried with SQL [11], [12] induced costly table joins. Thus, algebraic approaches based on tree-shaped patterns became popular for evaluating XML processing *natively* instead [13], [14]. Tree algebras indeed provide a formal framework for query expression and optimization, in a way similar to relational algebra with respect to the SQL language [15].

In this context, a tree pattern (TP), also called pattern tree or tree pattern query (TPQ) in the literature, models a user query over a data tree. Simply put, a tree pattern is a graphic representation that provides an easy and intuitive way of specifying the interesting parts from an input data tree that must appear in query output. More formally, a TP is *matched* against a tree-structured database to answer a query [16]. Figure 1 exemplifies this process. The upper left-hand side part of the figure features a simple XML document (a book catalog), and the lower left-hand side a sample XQuery that may be run against this document (and that returns the title and author of each book). The tree representations of the XML document and the associated query are featured on the upper and lower right-hand sides of Figure 1, respectively. At the tree level, answering the query translates in matching the TP against the data tree. This process can be optimized and outputs a data tree that is eventually translated back as an XML document.

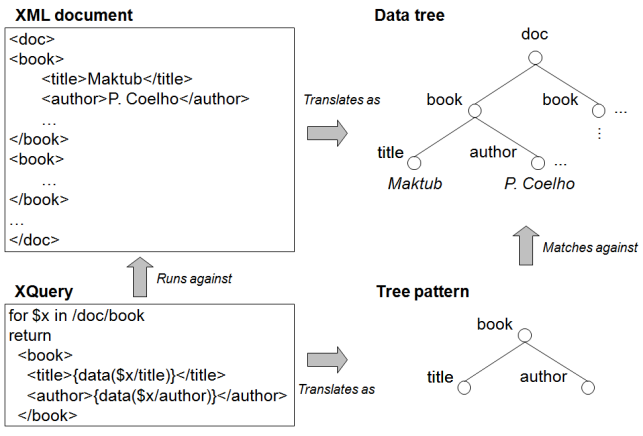

Fig. 1. Tree representation of XML documents and queries

Since the early 2000s, a tremendous amount of research has been based on, focusing on, or exploiting TPs for various purposes. However, few related reviews exist. Gou and Chirkova extensively survey querying techniques over persistently stored XML data [17]. Although the intersection between their

• *Marouane Hachicha and Jérôme Darmont are from the Université de Lyon (ERIC Lyon 2), 5 avenue Pierre Mendès-France, 69676 Bron Cedex, France.*
E-mails: marouane.hachicha@univ-lyon2.fr, jerome.darmont@univ-lyon2.fr

paper and ours is not empty, both papers are complementary. We do not address approaches related to the relational storage of XML data. By focusing on native XML query processing, we complement Gou and Chirkova's work with specificities such as TP structure, minimization approaches and sample applications. Moreover, we cover the many matching optimization techniques that have appeared since 2007. Other recent surveys are much shorter and focus on a particular issue, i.e., twig queries [18] and holistic matching [19].

The aim of this paper is thus to provide a global and synthetic overview of more than ten years of research about TPs and closely related issues. For this sake, we first formally define TPs and related concepts in Section 2. Then, we present and discuss various alternative TP structures in Section 3. Since the efficiency of TP matching against tree-structured data is central in TP usage, we review the two main families of TP matching optimization methods (namely, TP minimization and holistic matching approaches), as well as tangential but nonetheless interesting methods, in Section 4. Finally, we briefly illustrate the use of TPs through actual TP-based developments in Section 5. We conclude this paper and provide insight about future TP-related research in Section 6.

2 Background

In this section, we first formally define all the concepts used in this paper (Section 2.1). We also introduce a running example that illustrates throughout the paper how the TPs and related algorithms we survey operate (Section 2.2).

2.1 Definitions

2.1.1 XML document

XML is known to be a simple and very flexible text format. It is essentially employed to store and transfer text-type data. The content of an XML document is encapsulated within elements that are defined by tags [1]. These elements can be seen as a hierarchy organized in a tree-like structure.

2.1.2 XML fragment

An XML document may be considered as an ordered set of elements. Any element may contain subelements that may in turn contain subelements, etc. One particular element, which contains all the others, is the document's root. Any element (and its contents) different from the root is termed an XML fragment. An XML fragment may be modeled as a finite rooted, labeled and ordered tree.

2.1.3 Data tree

An XML document (or fragment) may be modeled as a data tree $t = (r, N, E)$, where N is a set of nodes, $r \in N$ is the root of t, and E is a set of edges stitching together couples of nodes $(n_i, n_j) \in N \times N$.

2.1.4 Data tree collection

An XML document considered as a set of fragments may be modeled as a data tree collection (also named forest in TAX [8]), which is itself a data tree.

2.1.5 Data subtree

Given a data tree $t = (r, N, E)$, $t^{'} = (r^{'}, N^{'}, E^{'})$ is a subtree of t if and only if:

1) $N^{'} \subseteq N$;
2) let $e^{'} \in E^{'}$ be an edge connecting two nodes $(n_i^{'}, n_j^{'})$ of $t^{'}$; there exists an edge $e \in E$ connecting two nodes (n_i, n_j) of t such that $n_i = n_i^{'}$ and $n_j = n_j^{'}$.

2.1.6 Tree pattern

A tree pattern p is a pair $p = (t, F)$ where:

1) $t = (r, N, E)$ is a data tree;
2) F is a formula that specifies constraints on t's nodes.

Basically, a TP captures a useful fragment of XPath [20]. Thus, it may be seen as the translation of a user query [21]. Translating an XML query plan into a TP is not a simple operation. Some XQueries are written with complex combinations of XPath and FLWOR expressions, and imply more than one TP. Thus, such queries must be broken up into several TPs. Only a single XPath expression can be translated into a single TP. The more complex a query is, the more its translation into TP(s) is difficult [10]. Starting from TPs to express user queries in a first stage, and optimize them in a second stage is a very effective solution used in XML query optimization [21].

2.1.7 Tree pattern matching

Matching a TP p against a data tree t is a function $f : p \rightarrow t$ that maps nodes of p to nodes of t such that:

- structural relationships are preserved, i.e., if nodes (x, y) are related in p through a parent-child relationship, denoted PC for short or by a simple edge / in XPath (respectively, an ancestor-descendant relationship, denoted AD for short or by a double edge // in XPath), their counterparts $(f(x), f(y))$ in t must be related through a PC (respectively, an AD) relationship too;
- formula F of p is satisfied.

The output of matching a TP against a data tree is termed a witness trees in TAX [8].

2.1.8 Tree pattern embedding

Embedding a TP p into a data tree t is a function $g : p \rightarrow t$ that maps each node of p to nodes of t such that structural relationships (PC and AD) are preserved.

The difference between embedding and matching is that embedding maps a TP against a data tree's *structure* only, whereas matching maps a TP against a data tree's structure *and contents* [22]. In the remainder of this paper, we use the more general term matching when referring to mapping TPs against data trees.

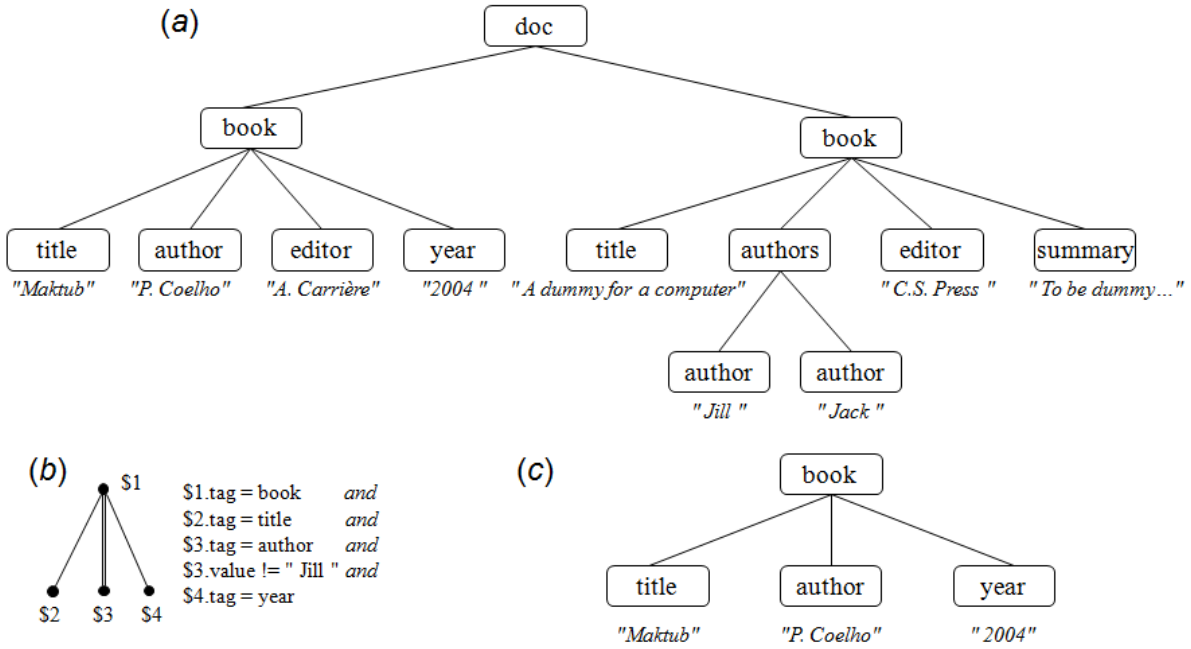

Fig. 2. Sample data tree (a), tree pattern (b) and witness tree (c)

2.1.9 Boolean tree pattern

A Boolean TP b yields a "true" (respectively "false") result when successfully (respectively unsuccessfully) matched against a data tree t. In other words, b has no output node(s) [23]: it yields a "true" result when t *structurally* matches (embeds) p.

2.2 Running example

Let us consider the data tree from Figure 2(a), which represents a collection of books. The root *doc* gathers *books*, each described by their *title*, *author(s)*, *editor*, *year* of publication and *summary*. Data tree nodes are connected by simple edges (/), i.e., PC relationships. Books do not necessarily bear the same structure. For instance, a summary is not present in all books, and some books are written by more than one author.

The TP from Figure 2(b) selects book titles, authors, and years. Moreover, formula F indicates that author must be different from Jill. Generally, a formula is a boolean combination of predicates on node types (e.g., $\$1.tag =$ book) and/or values (e.g., $\$3.value \; != $ "Jill"). Matching this TP against the data tree from Figure 2(a) outputs the data tree (or witness tree) from Figure 2(c). Only one book is selected from Figure 2(a), since the other one ($title =$ "A dummy for a computer"):

1) does not contain a year element;
2) is written by author Jill, which contradicts formula F.

Finally, the AD relationship $\$1//\$3$ in Figure 2(b)'s TP is correctly taken into account. It is not the structure of the book element with $title =$ "A dummy for a computer" that disqualifies it, but the fact that one of its authors is Jill. If this author was Gill, the book would be output.

3 Tree Pattern Structures

We review in this section the various TP structures found in the literature. Most have been proposed to support XML algebras (Section 3.1), but some have also been introduced for specific optimization purposes (Section 3.2). We conclude this section by discussing their features in Section 3.3.

3.1 Tree patterns in algebraic frameworks

The first XML algebras have appeared in 1999 [24], [25] in conjunction with efforts aiming to define a powerful XML query language [26], [27], [28], [29], [30]. Note that they have appeared before the first specification of XQuery, the now standard XML query language, which was issued in 2001 [7]. The aim of an XML tree algebra is to feature a set of operators to manipulate and query data trees. Query results are also data trees.

3.1.1 TAX tree pattern

The Tree Algebra for XML (TAX) [8] is one of the most popular XML algebras. TAX's TP preserves PC and AD relationships from an input ordered data tree in output, and satisfies a formula that is a boolean combination of predicates applicable to nodes. The example from Figure 2(b) corresponds to a TAX TP, save that node relationships are simple edges labeled AD or PC in TAX instead of being expressed as single or double edges. The TAX TP is the most basic TP

used in algebraic contexts. It has thus been greatly extended and enhanced.

3.1.2 Generalized tree pattern

The idea behind generalized tree patterns (GTPs) is to associate more options with TP edges in order to enrich matching. In TAX, one absent pattern node in the matched subtree prevents it to appear in output. A GTP extends the classical TAX TP by creating groups of nodes to facilitate their manipulation, and by enriching edges to be extracted by the *mandatory/optional* matching option [21].

Figure 3 shows an example of GTP, where the edge connecting the year element to its parent book node is dotted (i.e., optional), and title and author nodes are connected to the same parent book node with solid (i.e., mandatory) edges. This GTP permits to match all books described by their title and author(s), mandatorily, and that may be described by their year of publication. Matching it against the data tree from Figure 2(a) outputs both books ("Maktub" and "A dummy for a computer"), even though the second one does not contain a *year* element, while ensuring that the *title* and *author* elements exist in the matched book subtrees.

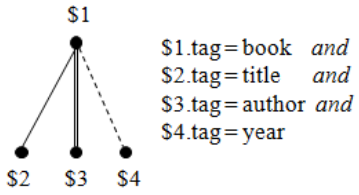

Fig. 3. Sample generalized tree pattern

Finally, note that the GTP from Figure 3, unlike the TP from Figure 2(b), does not include an $author$!= "Jill" clause in its formula. Retaining this predicate would not allow the second book ($title$ = "A dummy for a computer") to be matched by this GTP, since the boolean combination of formula elements (related with *and*) cannot be verified.

3.1.3 Annotated tree pattern

A feature, more than a limitation, of the TAX TP is that a set of subelements from the input data tree may all appear in the output data tree. For example, a TP with a single *author* node can match against a *book* subtree containing several *author* subelements. Annotated pattern trees (APTs) from the Tree Logical Class (TLC) algebra [31] solve this problem by associating matching specifications to tree edges. Matching options are:

- +: one to many matches;
- −: one match only;
- ⋆: zero to many matches;
- ?: zero or one match.

Figure 4 shows an example of APT where the − option is employed to extract books written by one author only. This APT matches only the first book from Figure 2(a) ($title$ = "Maktub"), since the second has two authors.

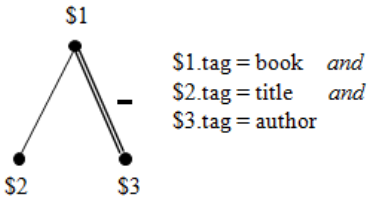

Fig. 4. Sample annotated pattern tree

3.1.4 Ordered annotated pattern tree

An APT, as a basic TAX TP, preserves the order output of nodes from the input data tree, whatever node order is specified in the TP. In other words, node order in witness trees is always the same as that of the input data tree. No reordering option is available, though it is essential when optimizing queries. To circumvent this problem, the APT used in TLC's `Select` and `Join` operators is supplemented with an order parameter (*ord*) based on an order specification (*O-Spec*) [13]. Four cases may occur:

1) *empty:* output node order is unspecified; *O-Spec* is empty;
2) *maintain:* input node order is retained; *O-Spec* duplicates input node order;
3) *list-resort:* nodes are wholly sorted with respect to *O-Spec*; input node order is forsaken;
4) *list-add:* nodes are partially sorted with respect to *O-Spec*.

For example, associating the *O-Spec* specification *[author, title]* to the APT from Figure 4 permits to select books ordered by author first, and then title. Graphically, the *author* node would simply appear on the left hand side of the witness tree and the *title* node on the right hand side.

3.2 Tree patterns used in optimization processes

Many TP matching optimization approaches extend the basic TP (Figure 2(b)) to allow a broader range of queries. In this section, we survey the TPs that introduce new, interesting features with respect to those already presented in Section 3.1.

3.2.1 Global query pattern tree

A global query pattern tree (G-QPT) is constructed from a set of possible ordered TPs proposed for the same query [32]. First, a root is created for the G-QPT. Then, each TP is merged with the G-QPT as follows:

- the TP root is merged with the G-QPT root;

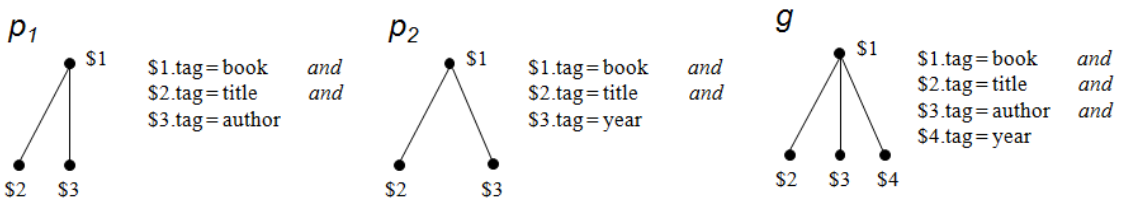

Fig. 5. Sample construction of global query pattern tree

- TP nodes are merged with G-QPT nodes with respect to node ordering and PC-AD relationships.

For example, TPs p_1 and p_2 from Figure 5 together answer query "select title, authors and year of publication of books" and merge into G-QPT g.

3.2.2 Twig pattern

Twig patterns are designed for directed and rooted *graphs* [33]. Here, XML documents are considered having a graph structure, thanks to ID references connecting nodes. Twig patterns respect PC-AD node relationships and express node constraints in formula F (called twig condition here) with operators such as *equals to*, *contains*, $\geq$, etc. In Figure 6, we represent the twig pattern that is equivalent to the TP from Figure 2(b).

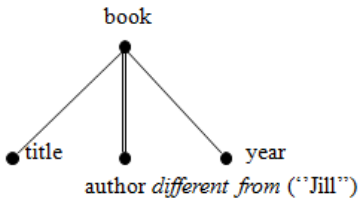

Fig. 6. Sample twig pattern

Moreover, distance-bounding twigs (DBTwigs) extend twig patterns to limit the large number of answers resulting from matching a classical twig pattern against a graph [33]. This method is called Filtering + Ranking. Its goal is to filter data to be matched by indicating the length of paths corresponding to descendant edges. DBTwigs also permit to indicate the number (0, 1...) of nodes to be matched. All these parameters are indicated in the graphical representation of DBTwigs.

3.2.3 Logical operator nodes

Izadi *et al.* include in their formal definition of TPs a set O of logical operator nodes [34]. $\wedge$, $\vee$ and $\oplus$ represent the binary AND, OR, and XOR logical operators, respectively. $\neg$ is the unary NOT operator. These operator nodes go further than GTPs' *mandatory/optional* edge annotations by specifying logical relationships between their subnodes. For instance, Figure 7 features a TP that selects book titles *and* either a set of authors *or* an editor.

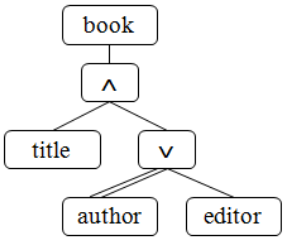

Fig. 7. Sample tree pattern with logical operator nodes

3.2.4 Node degree and output node specification

In a TP, the degree of a tree node x, denoted $degree(x)$, represents its number of children [35]. Miklau and Suciu define a maximal value for $degree(x)$ within a path expression optimization algorithm. This approach is mainly designed to check containment and equivalence of XPath fragments (Section 4.1.1). Then, translating XPath expressions into TPs requires the identification of an output node marked with a wildcard ($*$) in the TP. For example, suppose that we are only interested in book titles from Figure 2(a). Then, we would simply associate a $*$ symbol with the *title* node in the TP from Figure 2(b). Note that the other nodes must remain in the TP if they appear in formula F and thus help specify the output node.

3.2.5 Extended formula

Lakshmanan *et al.* further specify how formula F is expressed in a TP [22]. They define F as a combination of tag constraints (TCs), value-based constraints (VBCs) and node identity constraints (NICs). TCs specify constraints on tag names, e.g., $node.tag$ = "book". VBCs specify selection and join constraints on tag attributes and values using the operators $=$, $\neq$, $\leq$, $\geq$, $>$ and $<$. NICs finally determine whether two TP nodes are the same ($\approx$) or not. In addition to TCs, VBCs and NICs, wildcards ($*$) are associated with untagged nodes.

For example, in Figure 8, we extend the TP from Figure 2(b) with a VBC indicating that books to be selected must be of type "Novel". We suppose here that a type is associated with each *book* node, that the first book ($title$ = "Maktub") is of type "Novel", and that the second book ($title$ = "A dummy for a computer") is of type "Technical book". Then, the output tree would of course only include the book entitled "Maktub".

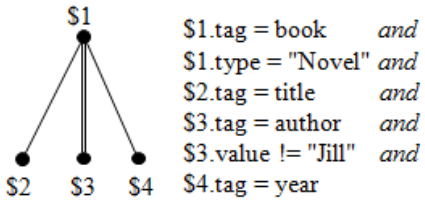

Fig. 8. Sample tree pattern with a value-based constraint

3.2.6 Extended tree pattern

Extended TPs complement classical TPs with a negation function, wildcards and an order restriction [36]. For example, in Figure 9, the negative function (denoted $\neg$) helps specify that we look for an edited book, i.e., with no author node. The wildcard node $*$ can match any single node in a data tree. Note that the wildcard has a different meaning here than in Section 3.2.4, where it denotes an output node, while output node names are underlined in extended TPs. Finally, the order restriction denoted by a $<$ in a box means that children of node book are ordered, i.e., title must come before the $*$ node.

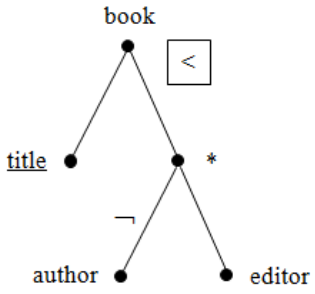

Fig. 9. Sample extended tree pattern

3.3 Discussion

In this section, we discuss and compare the TPs surveyed in Sections 3.1 and 3.2. For this sake, we introduce four comparison criteria.

- *Matching power:* Matching encompasses two dimensions. Structural matching guarantees that only subtrees of the input data tree that map the TP are output. Matching by value is verifying formula F. We mean by matching power all the matching options (edge annotations, logical operator nodes, formula extensions...) beyond these basics. Improving matching power helps filter data more precisely.
- *Node reordering capability:* Order is important in XML querying, thus modern TPs should be able to alter it [13]. We mean by node reordering capability the ability of a TP to modify output node order when matching against any data tree. Note that node reordering could be classified as a matching capability, but the importance of ordering witness trees leads us to consider it separately.
- *Expressiveness:* Expressiveness states how a logical TP, i.e., a TP under its schematic form, translates into the corresponding physical implementation. The physical form of a TP may be either expressed with an XML query language (XQuery, generally), or an implementation within an XML database offering query possibilities. Using a TP for other purposes but querying, e.g., in optimization algorithms, is not accounted toward expressiveness. Note that a TP that cannot be expressed in physical form is usually considered useless.
- *Supported optimizations:* TPs are an essential element of XML querying [16], [37]. Hence, many optimization approaches translate XML queries into TPs, optimize them, and then translate them back into optimized queries. Optimizing a TP increases its matching power. This criterion references the different kinds of optimizations supported by a given TP.

TP comparison with respect to matching power, node reordering capability, expressiveness and supported optimizations follows (Sections 3.3.1, 3.3.2, 3.3.3 and 3.3.4, respectively), and is synthesized in Section 3.3.5.

3.3.1 Matching power

The most basic TP is TAX's [8]. Thus, matching a TAX TP against a data tree collection is very simple: it is only based on tree structure and node values specified in formula F. If F is absent, a TAX TP matches all possible subtrees from the input data tree.

Associating a *mandatory/optional* status to GTP edges [21] increases the number of matched subtrees in output. Only one absent edge in a TAX TP with respect to a subtree containing all the other edges in this TP prevents matching, while the candidate subtree is "almost perfect". If this edge is labeled as *optional* in a GTP, matching becomes successful. Even more flexibility is achieved by logical operator nodes [34], which allow powerful matching options, especially when logical operators are combined.

With APTs, matching precision is further improved through edge annotations [31]. Annotations in APTs force the matching process to extract data according to their nature. For example, a classical TAX TP such as the one from Figure 2(b), extracting books by titles and authors, does not allow controlling the number of authors. APTs help select only books with, e.g., more than one author, as in the example from Figure 4. Unfortunately, annotations only allow two maximum cardinalities: one or several. DBTwigs [33] complement APTs in this respect, by allowing to choose the exact number of nodes to be matched. Node degree

(number of children) [35] or the negation function of extended TPs [36] can also be exploited for this sake.

Finally, up to now, we focused on TP structure because few differences exist in formulas. Graphically, twig patterns associate constraints with nodes directly in the schema [33] while in the TAX TP and its derivatives, they appear in formula F, but this is purely cosmetic. Only TCs, VBCs and NICs help further structure formula F [22].

3.3.2 Node reordering capability

Despite many studies model XML documents as unordered data trees, order *is* essential to XML querying [13]. In XQuery, users can (re)order output nodes simply through the `Order by` clause inherited from SQL [15]. Hence, modern TPs should include ordering features [16], [37].

However, among TAX TP derivatives, only ordered APTs and extended TPs feature reordering features. In G-QTPs, preorders associated with ordered TP nodes help determine output order [32]. Unfortunately, order is disregarded in all other TP proposals.

3.3.3 Expressiveness

TAX TPs and their derivatives (GTPs and APTs) do not translate into an XML query language, but they are implemented, through the TLC physical algebra [31], in the TIMBER XML database management system [38]. TIMBER permits to store XML in native format and offers a query interface supporting both classical XQuery fragments and TAX operators. Note that TAX operators include a `Group by` construct that has no equivalent in XQuery.

Translating TAX TPs for XML querying follows nine steps:

1) identify all TP elements in the `FOR` clause;
2) push formula F's predicates into the `WHERE` clause;
3) eliminate duplicates with the help of the `DISTINCT` keyword;
4) evaluate aggregate expressions in `LET` clauses;
5) indicate tree variables to be joined (join conditions) via the `WHERE` clause;
6) enforce any remaining constraint in the `WHERE` clause;
7) evaluate `RETURN` aggregates;
8) order output nodes with the help of the `ORDER BY` clause;
9) project on the elements indicated in the `RETURN` clause.

Similarly, Lakshmanan *et al.* test the satisfiability of TPs translated from XPath expressions and XQueries, and then express them back in XQuery and evaluate them within the XQEngine XQuery engine [39]. The other TPs we survey are used in various algorithms (containment and equivalence testing, TP rewriting, frequent TP mining...). Hence, their expressiveness is not assessed.

3.3.4 Supported optimizations

Approaches purposely proposing TPs (i.e., algebraic approaches) are much fewer than approaches using TPs to optimize XML querying. Moreover, even algebraic approaches such as TAX do address optimization issues. Since XML queries are easy to model with TPs, researchers casually translate any XML query in TPs, which are then optimized and implemented in algorithms, XML queries or any other optimization framework.

Hence, the biggest interest of the TP community lies in enriching and optimizing matching. Matching opportunities offered by TAX TPs, optional edges of GTPs, annotations, ordering specification and duplicate elimination of APTs, and extended TPs aim to achieve more results and/or better precision. GTP and APT matching characteristics prove their efficiency in TIMBER [38].

Finally, satisfiability is an issue related to containment, a concept used in minimization approaches. Satisfiability means that there exists a database, consistent with the schema if one is available, on which the user query, represented by a TP, has a non-empty answer [22], [40], [41], [42].

3.3.5 Synthesis

We recapitulate in Table 1 the characteristics of all TPs surveyed in this paper with respect to our comparison criteria.

In summary, matching options in TPs are numerous and it would probably not be easy to set up a metric to rank them. The best choice, then, seems to select a TP variant that is adapted to the user's or designer's needs. However, designing a TP that pulls together most of the options we survey is certainly desirable to maximize matching power.

With respect to output node ordering, we consider the *ord* order parameter introduced in APTs the simplest and most efficient approach. A list of ordered elements can indeed be associated with any TP, whatever the nature of the input data tree (ordered or unordered). However, it would be interesting to generalize the *ord* specification to other, possibly more complex, operators beside `Select` and `Join`, the two only operators benefiting from ordering in APTs.

Expressiveness is a complex issue. Translating XML queries into TPs is indeed easier than translating TPs back into an XML query plan. XQuery, although the standard XML query language, suffers from limitations such as the lack of a `Group by` construct. Thus, it is more efficient to implement TPs and exploit them to enrich XML querying in an ad-hoc environment such as TIMBER's. We think that the richer the pattern, with matching options, ordering specifications, possibility to associate with many operators (and other options if possible), the more efficient querying is, in terms of user need satisfaction.

TABLE 1
Synthesis of tree pattern characteristics

TP	Matching features	Reordering	Expressiveness	Supported optimizations
TAX TP [8]	Basic	No	TIMBER	Matching
GTP [21]	Mandatory/optional edges	No	TIMBER	Matching
APT [31]	Edge cardinality	No	TIMBER	Matching
Ordered APT [13]	Order specification	Yes	TIMBER	Matching
G-QPT [32]	Set of TPs	Yes	N/A	Unspecified
Twig pattern [33]	Graph structure	No	N/A	XML graph querying
DBTwig [33]	Filtering + Ranking + Matching cardinality	No	N/A	XML graph querying
Izadi *et al.*'s TP [34]	Logical operator nodes + Potential target node	No	N/A	Matching
Miklau and Suciu's TP [35]	Node degree + Output node	No	N/A	Containment and equivalence testing
Lakshmanan *et al.*'s TP [22]	Tag constraints + Value-based constraints + Node identity constraints	No	XQuery	Satisfiability testing
Extended TP [36]	Negation function + Wildcards	Yes	N/A	Matching

Finally, minimization, relaxation, containment, equivalence and satisfiability issues lead the TP community to optimize these tasks. However, most TPs used in this context are basic, unlike TPs targeted at matching optimization, which allows optimizing XML queries wherein are translated optimized TPs. In short, the best-optimized TP must be minimal, satisfiable, and offer as many matching options as possible.

4 TREE PATTERN MATCHING OPTIMIZATION

THE aim of TPs is not only to provide a graphical representation of queries over tree-structured data, but also and primarily, to allow matching queries against data trees. Hence, properly optimizing matching is primordial to achieve good query response time.

In this section, we present the two main families of approaches for optimizing matching, namely minimization methods (Section 4.1) and holistic matching approaches (Section 4.2). We also survey a couple of other specific approaches (Section 4.3), before globally discussing and comparing all TP matching optimization methods (Section 4.4).

4.1 Tree pattern minimization

The efficiency of TP matching depends a lot on the size of the pattern [16], [20], [43], [44]. It is thus essential to identify and eliminate redundant nodes in the pattern and do so as efficiently as possible [16]. This process is called TP minimization.

All research related to TP minimization is based on a pioneer paper by Amer-Yahia *et al.* [16], who formulate the problem as follows: given a TP, find an equivalent TP of the smallest size. Formally, given a data tree t and a TP p of size (i.e., number of nodes) n, let $S = \{p_i\}$ be the set of TPs of size n_i contained in p ($p_i \subseteq p$ and $n_i \leq n\ \forall i$). Minimizing p is finding a TP $p_{min} \in S$ of size n_{min} such that:

- $p_{min} \equiv p$ when matched against t;
- $n_{min} \leq n_i\ \forall i$.

Moreover, a set C of integrity constraints (ICs) may be integrated into the problem. There are two forms of ICs:

1) each node of type A (e.g., book) must have a child (respectively, descendant) of type B (e.g., author), denoted $A \rightarrow B$ (respectively, $A \Rightarrow B$);
2) each node of type A (e.g., book) must have a descendant of type C (e.g., name), knowing that C is a descendant of B (e.g, author), i.e., $A \Rightarrow C$ knowing that $B \Rightarrow C$.

Since TP minimization relies on the concepts of containment and equivalence, we first detail how containment and equivalence are tested (Section 4.1.1). Then, we review the approaches that address the TP minimization problem without taking ICs into account (Section 4.1.2), and the approaches that do (Section 4.1.3).

4.1.1 Containment and equivalence testing

Let us first further formalize the definitions of containment and equivalence. *Containment* of two TPs p_1 and p_2 is defined as a node mapping relationship $h : p_1 \rightarrow p_2$ such that [16], [35]:

- h preserves node type, i.e., $\forall x \in p_1$, x and $h(x)$ must be of the same type. Moreover, if x is an output node, $h(x)$ must be an output node too;
- h preserves node relationships, i.e., if two nodes (x, y) are linked through a PC (respectively, AD)

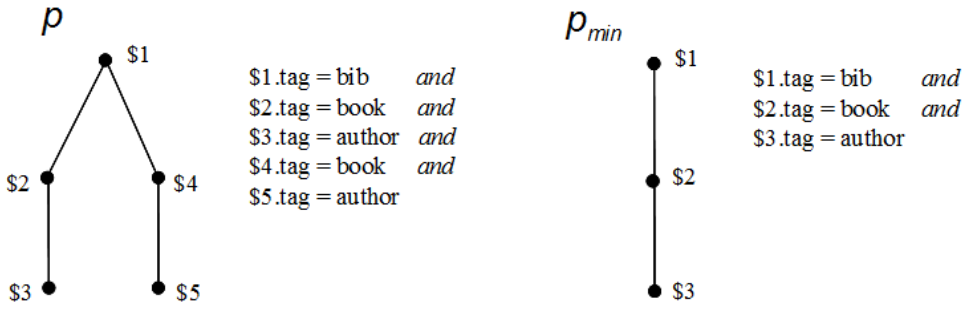

Fig. 10. Simple minimization example

relationship in p_1, $(f(x), f(y))$ must also be linked through a PC (respectively, AD) relationship in p_2.

Note that function h is very similar to function f that matches a TP to a data tree (Section 2.1.7), but here, h is a homomorphism between two TPs [45]. Moreover, containment may be tested between a TP fragment and this TP as a whole. h is then an endomorphism.

Finally, *equivalence* between two TPs is simply considered as two-way containment [16], [20]. Formally, let p_1 and p_2 be two TPs. $p_1 \equiv p_2$ if and only if $p_1 \subseteq p_2$ and $p_2 \subseteq p_1$.

Miklau and Suciu show that containment testing is intractable in the general case [35]. They nonetheless propose an efficient algorithm for significant particular cases, namely TPs with AD edges and an output node. This algorithm is based on tree automata, the objective being to reduce the TP containment problem to that of regular tree languages. In summary, given two TPs p_1 and p_2, to test whether $p_1 \subseteq p_2$:

- p_1's nodes and edges are matched to deterministic finite tree automaton A_1's states and transitions, respectively;
- p_2's nodes and edges are matched to alternating finite tree automaton A_2's states and transitions, respectively;
- if $lang(A_1) \subseteq lang(A_2)$, then $p_1 \subseteq p_2$, where $lang(A_i)$ is the language associated with automaton A_i $(i = \{1, 2\})$.

Similar approaches further test TP containment under Document Type Definition (DTD) constraints [46], [47]. For instance, Wood exploits regular tree grammars (RTGs) to achieve containment testing [47]. Let D be a DTD and G_1 and G_2 RTGs corresponding to p_1 and p_2, respectively. Then $p_1 \subseteq p_2$ if and only if $(D \cap G_1) \subseteq (D \cap G_2)$.

4.1.2 Unconstrained minimization

The first TP minimization algorithm, Constraint Independent Minimization (CIM) [16], eliminates redundant nodes from TP p by exploiting the concept of images. Let there be a node $x \in p$. Its list of images, denoted $images(x)$, is composed of nodes from p that bear the same type as x, but are different from x. For each leaf $x \in p$, CIM searches for nodes of the same type as x in the set of descendants of $images(parent(x))$, where $parent(x)$ is x's parent node in p. If such nodes are found, x is redundant and thus deleted from p. CIM then proceeds similarly, in a bottom-up fashion, on $parent(x)$, until all nodes in p (except output nodes that must always be retained) have been checked for redundancy.

For example, let us consider TP p from Figure 10. Without regarding node order, let us check leaf node \$5 for redundancy. $parent(\$5) = \$4, images(\$4) = \{\$2\}, descendants(\$2) = \{\$3\}$, where $descendants(x)$ is the set of descendants of node x. \$3 bears the same type as \$5 (author), thus \$5 is deleted. Similarly, \$4 is then found redundant and deleted, outputting p_{min} in Figure 10. Further testing \$1, \$2 and \$3 does not detect any new redundant node. Moreover, deleting another node from p_{min} would break the equivalence between p and p_{min}. Hence, p_{min} is indeed minimal.

All minimization algorithms subsequent to CIM retain its principle while optimizing complexity. One strategy is to replace images by more efficient relationships, i.e., coverage [48], [49], also called simulation [43]. Let p be the TP to minimize, and $x, y \in p$ two of its nodes. If y covers x ($x \angle y$) [50]:

- the types of x and y must be identical;
- if x has a child (respectively descendant) node x', y must have a child (respectively descendant) node y' such that $x' \angle y'$.

$cov(x)$ denotes the set of nodes that cover x. $cov(x) = x$ if x is an output node. Then, the minimization process tests node redundancy in a top-down fashion as follows: $\forall x \in p$, $\forall x' \in children(x)$ (respectively $descendants(x)$), if $\exists x'' \in children(x)$ (respectively $descendants(x)$) such that $x'' \in cov(x')$, then x' *and the subtree rooted in* x' are deleted. $children(x)$ (respectively $descendants(x)$) is the set of direct children (respectively, all descendants) of node x.

Another strategy is to simply prune subtrees recursively [51]. The subtree rooted at node x, denoted $subtree(x)$, is minimized in two steps:

1) $\forall x', x'' \in children(x)$, if $subtree(x') \subseteq subtree(x'')$ then delete $subtree(x')$;

2) $\forall x' \in children(x)$ (remaining children of x), minimize $subtree(x')$.

A variant proceeds similarly, by first searching in a TP p for any subtree p_i redundant with sp, where sp is p stripped of its root [20]. Formally, the algorithm tests whether $p - sp_i \subseteq p_i$. Redundant subtrees are removed. Then, the algorithm is recursively executed on unpruned subtrees sp_i.

4.1.3 Minimization under integrity constraints

Taking ICs into account in the minimization process is casually achieved as follows.

1) Augment the TP to minimize with nodes and edges that represent ICs. This is casually achieved with the classical chase technique [52]. For instance, if we had a book $\rightarrow$ author IC (nodes of type "book" must have a child of type "author"), we would add one child node of type author to each of the nodes \$2 and \$4 from Figure 10. Note that augmentation must only apply to nodes from the original TP, and not to nodes previously added by the chase.
2) Run any TP minimization algorithm, without testing utilitarian nodes introduced in step #1 for redundancy, so that ICs hold.
3) Delete utilitarian nodes introduced in step #1.

This process has been applied onto the CIM algorithm, to produce ACIM (Augmented CIM) [16], as well as on its coverage-based variants [43], [48], [53]. Since the size of an augmented TP can be much larger than that of the original TP, an algorithm called Constraint Dependant Minimization (CDM) also helps identify and prune all TP nodes that are redundant under ICs [16]. CDM actually acts as a filter before ACIM is applied. CDM considers a TP leaf x' of type T' redundant and removes it if one of the following conditions holds.

- $parent(x') = x$ (respectively $ancestor(x') = x$) of type T and there exists an IC $T \rightarrow T'$ (respectively $T \Rightarrow T'$).
- $parent(x') = x$ (respectively $ancestor(x') = x$) of type T, $\exists x''/parent(x'') = x$ (respectively $ancestor(x'') = x$) of type T'', and there exists an IC $T' = T''$ (respectively, there exists one of the ICs $T'' \Rightarrow T'$ or $T' = T''$).

An alternative simulation-based minimization algorithm also includes a similar prefilter, along with an augmentation phase that adds to the nodes of $cov(x)$ their ancestors instead of using the chase [43].

Finally, the scope of ICs has recently been extended to include not only forward and subtype (FT) constraints (as defined in Section 4.1), but also backward and sibling (BS) constraints [44]:

- each node of type A (e.g., author) must have a parent (respectively, ancestor) of type B (e.g., book), i.e., $A \leftarrow B$ (respectively, $A \Leftarrow B$);
- each node of type A (e.g., book) that has a child of type B (e.g., editor) must also have a child of type C (e.g, address), i.e., if $A \rightarrow B$ then $A \rightarrow C$.

Under FBST constraints, several minimal TPs can be achieved (*vs.* one only under FT constraints), which allows further optimizations of the chase augmentation and simulation-based minimization processes.

4.2 Holistic tree pattern matching

While TP minimization approaches (Section 4.1) wholly focus on the TP side of the matching process, holistic matching (also called holistic twig join [19]) algorithms mainly operate on minimizing access to the input data tree when performing actual matching operations. The initial binary join-based approach for matching proposed by Al-Khalifa *et al.* [54] indeed produces large intermediate results. Holistic approaches casually optimize TP matching in two steps [55]:

1) *labeling:* assign to each node x in the data tree t an integer label $label(x)$ that captures the structure of t (Section 4.2.1);
2) *computing:* exploit labels to match a twig pattern p against t without traversing t again (Section 4.2.2).

Moreover, recent proposals aim at reducing data tree size by exploiting structural summaries in combination to labeling schemes. We review them in Section 4.2.3.

Let us finally highlight that, given the tremendous number of holistic matching algorithms proposed in the literature, it is quite impossible to review them all. Hence, we aim in the following sections at presenting the most influential. The interested reader may further refer to Grimsmo and Bjørklund's survey [19], which uniquely focuses on and introduces a nice history of holistic approaches.

4.2.1 Labeling phase

The aim of data tree labeling schemes is to determine the relationship (i.e., PC or AD) between two nodes of a tree from their labels alone [55]. Many labeling schemes have been proposed in the literature. We particularly focus in this section on the region encoding (or containment) and the Dewey ID (or prefix) labeling schemes that are used in holistic approaches. However, other approaches do exist, based on a tree-traversal order [56], prime numbers [57] or a combination of structural index and inverted lists [58], for instance.

The region encoding scheme [59] labels each node x in a data tree t with a 3-tuple $(start, end, level)$, where $start$ (respectively, end) is a counter from the root of t until the start (respectively, the end) of x (in depth first), and $level$ is the depth of x in t. For example, the data tree from Figure 2(a) is labeled in Figure 11, with the region encoding scheme label indicated between

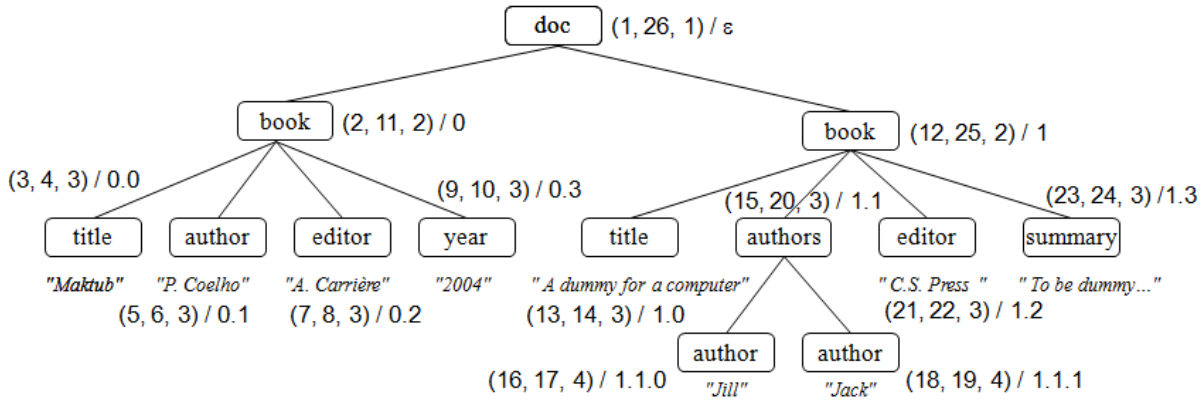

Fig. 11. Sample data tree labeling

parentheses. In Figure 11, node $x(title =$ "Maktub") is labeled (3, 4, 3). $start = 3$ because x is the first child of the node with $start = 2$; $end = 4$ because x has no children (thus $end = start + 1$); and $level = 3$ (level 1 being the root's).

Now, let x and x' be two nodes labeled (S, E, L) and (S', E', L'), respectively. Then:

- x' is a descendant of x if and only if $S < S'$ and $E' < E$;
- x' is a child of x if and only if $S < S'$, $E' < E$ and $L' = L + 1$.

For example, in Figure 11, node $(author =$ "Jack") labeled (18, 19, 4) is a descendant from the node $book$ labeled (12, 25, 2) and a child of node $authors$ labeled (15, 20, 3).

The Dewey ID scheme [60] labels tree nodes as a sequence. The root node is labeled ϵ (empty). Its children are labeled 0, 1, 2, etc. Then, at any subsequent level, the children of node x are labeled $label(x).0$, $label(x).1$, $label(x).2$, etc. More formally, for each non-root element x', $label(x') = label(x).i$, where x' is the i^{th} child of x. Thus, each label embeds all ancestors of the associated node. For example, in Figure 11, the Dewey ID label is featured on the right hand side of the "/", for each node. Node $(author =$ "Jack"), labeled 1.1.1, is the second child of the node labeled 1.1 (i.e., $authors$), and a descendant of the node labeled 1 (i.e., the right hand side $book$).

The Dewey ID scheme has been extended to incorporate node names [61], by exploiting schema information available in a DTD or XML schema. Encoding node names along a path into a Dewey label provides not only the labels of the ancestors of a given node, but also their names. Moreover, the Dewey ID scheme suffers from a high relabeling cost for dynamic XML documents where nodes can be arbitrarily inserted and deleted. Thus, variant schemes, namely ORDPATH [62] and Dynamic DEwey (DDE) [63], have been devised to dynamically extend the domain of label component values, so that no global relabeling is required.

The main difference between the region encoding and Dewey ID labeling schemes lies in the way structural relationships can be inferred from a label. While region encoding necessitates two nodes to determine whether they are related by a PC or AD relationship, Dewey IDs directly relate to ancestors and thus only require to know the current node's label. A Dewey ID-labeled data tree is also easier to update than a region encoded data tree [64].

4.2.2 Computing phase

Various holistic algorithms actually achieve TP matching, but they all exploit a data list that, for each node, contains all labels of nodes of the same type. In this section, we first review the approaches based on the region encoding scheme, which were first proposed, and then the approaches based on the Dewey ID scheme.

As their successors, the first popular holistic matching algorithms, PathStack and TwigStack, proceed in two steps [59]: intermediate path solutions are output to match each query root-to-leaf path, and then merged to obtain the final result. For example, let us consider the TP represented on the left hand side of Figure 12, to be matched against the data tree from Figure 11. Intermediate path solutions follow, expressed as labels.

- $book/title$: (2, 11, 2) (3, 4, 3), (12, 25, 2) (13, 14, 3)
- $book/editor$: (2, 11, 2) (7, 8, 3), (12, 25, 2) (21, 22, 3)

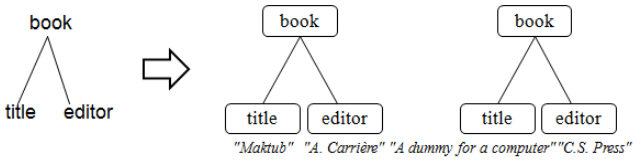

Fig. 12. Sample holistic matching

After merging these intermediate paths, we obtain the label paths below, which correspond to the witness trees represented on the right hand side of Figure 12.

- (2, 11, 2) (3, 4, 3) (7, 8, 3)
- (12, 25, 2) (13, 14, 3) (21, 22, 3)

One issue with TwigStack is that it only considers AD relationships in the TP and does not consider level information. Hence, it may output many useless intermediate results for queries with PC relationships. Moreover, it cannot process queries with order predicates.

On one hand, in order to reduce the search space, path summaries that exploit schema information may be used [65]. If a pattern subtree matches a data tree several times, TwigStack loads streams for all distinct paths, whether these streams contribute to the output or not. Path summaries help distinguish whether the occurrences of each pattern subtree are output elements or not. Those that do not are pruned.

On the other hand, TwigStackList better controls the size of intermediate results by buffering parent elements in PC relationships in a main-memory list structure [66]. Thus, only AD relationships in branching edges[1] are handled by TwigStackList, and not in all edges as with TwigStack. OrderedTJ builds upon TwigStackList by handling order specifications in TPs [67]. OrderedTJ additionally checks the order conditions of nodes before outputting intermediate paths, with the help of a stack data structure.

Since TwigStack and OrderedTJ partition data to streams according to their names alone, two new data streaming techniques are introduced in iTwigJoin [68]: the tag+level and prefix path schemes. In the OrderedTJ algorithm, only AD relationships in branching edges are taken into account. The tag+level scheme also takes PC relationships in all edges into account. The prefix path scheme further takes 1-branching into account.

An eventual enhancement has been brought by Twig^{2}Stack, which optimizes TwigStack by further reducing the size of intermediate results [69]. Twig^{2}Stack associates each query node x with a hierarchical stack. A node x' is pushed into hierarchical stack $HS[x]$ if and only if x' satisfies the subtwig query rooted at x. Matching can be determined when an entire sub-tree of x' is seen with respect to post-order data tree traversal. Baca *et al.* also fuse TwigStack and Twig^{2}Stack along the same line, still to reduce the size of intermediate results [70].

Simply replacing the region encoding labeling scheme by the Dewey ID scheme would not particularly improve holistic matching approaches, since they would also need to read labels for all tree nodes. However, exploiting the *extended* Dewey labeling scheme allows further improvements.

TJFast constructs, for each node x in the TP, an input stream T_x [61]. T_x contains the ordered extended Dewey labels of nodes of the same type as x. As TwigStackList, TJFast assigns, for each branching node b, a set of nodes S_b that are potentially query answers. But with TJFast, the size S_b is always bounded by the depth of the data tree. TJFast+L further extends TJFast by including the tag+level streaming scheme [64].

Eventually, Lu *et al.* have recently identified a key issue in holistic algorithms, called matching cross [36]. If a matching cross is encountered, a holistic algorithm either outputs useless intermediate results and becomes suboptimal, or misses useful results and looses its matching power. Based on this observation, the authors designed a set of algorithms, collectively called TreeMatch, which use a concise encoding to present matching results, thus reducing useless intermediate results. TreeMatch has been proved optimal for several classes of queries based on extended TPs (Section 3.2.6).

4.2.3 Structural summary-based approaches

These approaches aim at avoiding repeated access to the input data tree. Thus, they exploit structural summaries similar to the DataGuide proposed for semi-structured documents [71]. A DataGuide's structure describes using one single label all the nodes whose labels (types) are identical. Its definition is based on targeted path sets, i.e., sets of nodes that are reached by traversing a given path. For example, the DataGuide corresponding to the data tree from Figure 2(a) is represented in Figure 13.

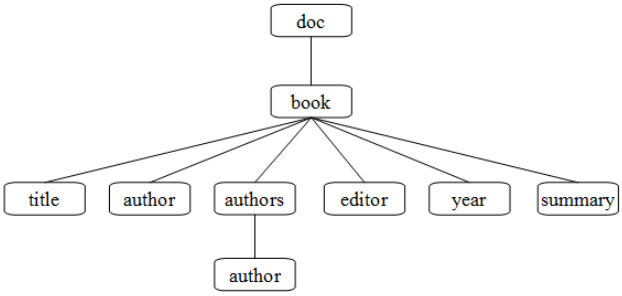

Fig. 13. Sample DataGuide

While a DataGuide can efficiently answer queries with PC edges (by matching the query path against the label path directly), it cannot process queries with AD edges nor twig queries, because it does not preserve hierarchical relationships [72]. Combining a DataGuide with a labeling scheme that captures AD relationships allows the reconstruction, for any node x in a data tree t, of the specific path instance x belongs to. Thus, in the computing phase, node labels can be compared without accessing labels related to inner TP nodes [34].

For instance, TwigX-Guide combines a DataGuide to the region encoding labeling scheme [72]. Version Tree is an annotated DataGuide in which labels include a version number for nodes of the same type

1. A *branching node* is a node whose number of children is greater than one. All edges originating from a branching node are called *branching edges* [66].

(e.g., all $book$ nodes) [73]. Combined to the Dewey ID labeling scheme, it supports a matching algorithm called TwigVersion. Finally, QueryGuide labels DataGuide nodes with Dewey ID lists and is part of the S^3 matching method [34]. All three approaches have been experimentally shown to perform matching faster than previous holistic algorithm such as TwigStack and TJFast.

4.3 Other pattern tree matching approaches

We present in this section matching approaches of interest that do not fall into the minimization and holistic families of methods, namely tree homeomorphism matching (Section 4.3.1) and TP relaxation (Section 4.3.2).

4.3.1 Tree homeomorphism matching

The tree homeomorphism matching problem is a particular case of the TP matching problem. More precisely, the considered TPs only bear descendant edges. Formally, given a TP p and a data tree t, tree homeomorphism matching aims at determining whether there is a mapping θ from the nodes of p to the nodes t such that if node x' is a *child* of x in p, then $\theta(x')$ is a *descendant* of $\theta(x)$ in t.

Götz *et al.* propose a series of algorithms that aim at reducing the time and space complexity of previous homeomorphism matching algorithms [74]. Their whole work is based on a simple matching procedure called MATCH. Let x be a node of TP p and y a node of data tree t. MATCH tests whether the subtree rooted at x, $subtree(x)$, matches $subtree(y)$. If y matches x, children of x are recursively tested to match any child of y. If y does not match x, then x is recursively tested to match any child of y. MATCH uses the recursion stack to determine which function call to issue next or which final value to return, e.g., to determine the data node y onto which x's parent was matched in t before proceeding with x's siblings. In opposition, L-MATCH recomputes the information necessary to make the decision with a backtracking function.

MATCH and L-MATCH are space-efficient top-down algorithms, but involve a lot of recomputing and thus bear a high time complexity. Thus, Götz *et al.* also introduce a bottom-up strategy. It is based on algorithm TMATCH that addresses the tree homeomorphism problem. TMATCH exploits a left-to-right post-order ordering $<_{post}$ on nodes, and returns the largest (w.r.t. $<_{post}$) TP node x in an interval $[x_{from}, x_{until}]$ (still w.r.t. $<_{post}$) such that $subtree(y)$ matches $[x_{from}, x]$ if x exists; and $x_{from} - 1$ (the predecessor of x_{from} w.r.t. $<_{post}$) otherwise. Finally, TMATCH-ALL generalizes TMATCH to address the tree homeomorphism matching problem, i.e., TMATCH-ALL computes all possible exact matches of p against t.

4.3.2 Tree pattern relaxation

TP relaxation is not an optimization of the matching process *per se*, but an optimization of its result with respect to user expectations. TP relaxation indeed allows approximate TP matching and returns ranked answers in the spirit of Information Retrieval [37]. Four TP relaxations are proposed, the first two relating to structure and the last two to content.

1) *Edge generalization* permits a PC edge in the TP to be generalized to an AD edge. For example, in Figure 14, the $book/title$ edge can be generalized to allow books with any descendant $title$ node to appear in the witness tree.
2) *Subtree promotion* permits to connect a whole subtree to its grandparent by an AD edge. For example, in Figure 14, the $address$ node can be promoted to allow $book$ nodes that have a descendant $address$ node to be output even if the $address$ node is not a descendant of the $editor$ node.
3) *Leaf node deletion* permits a leaf node to be deleted. For example, in Figure 14, the $summary$ node can be deleted, allowing for books to appear in the witness tree whether they bear a summary or not.
4) *Node generalization* permits to generalize the type of a query node to a supertype. For example, in Figure 14, the node $book$ could be generalized to node doc (document) from Figure 2(a).

Answer ranking is achieved by computing a score. To this aim, the weighted TP is introduced, where each node and edge is associated with an exact weight ew and a relaxed weight rw such that $ew \geq rw$. Figure 14 features a sample weighted TP. In this example, the score of exact matches of the weighted TP is equal to the sum of the exact weight of its nodes and edges, i.e., 41. If the node $book$ was generalized to doc, the score of an approximate answer that is a document is the sum of $rw(book)$ and the exact weight of the other nodes and edges, i.e., 35.

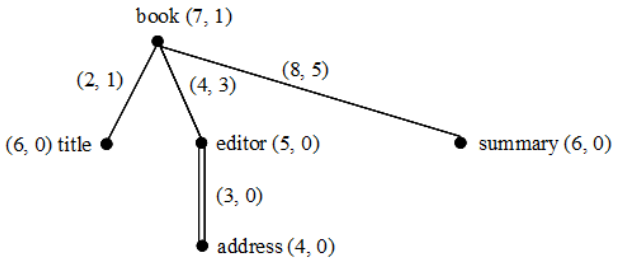

Fig. 14. Sample weighted tree pattern

4.4 Discussion

In this section, we discuss and compare the TP matching optimization approaches surveyed in Sections 4.1, 4.2 and 4.3; except TP relaxation, whose goal

is quite different (i.e., augmenting user satisfaction rather than matching efficiency). Choosing objective comparison criteria is pretty straightforward for algorithms. *Time* and *space complexity* immediately come to mind, though they are diversely documented in the papers we survey.

We could also retain algorithm *correctness* as a comparison criterion, although proofs are quite systematically provided by authors of matching optimization approaches. Thus, we only notice here that none of the minimization algorithms reviewed in Section 4.1 actually test the equivalence of minimal TP p_{min} to original TP p. More precisely, containment (i.e., $p_{min} \subseteq p$) is checked *a posteriori*, but the minimization process being assumed correct, equivalence is not double-checked.

Algorithm comparison with respect to complexity follows (Sections 4.4.1 and 4.4.2), and is synthesized in Section 4.4.3, where we also further discuss the complementarity between TP minimization and holistic approaches.

4.4.1 Time complexity

Time complexity is quite well documented for minimization approaches. Except the first, naive matching algorithms [16], all optimized minimization algorithms, whether they take ICs into account or not, have a worst-case time complexity of $O(n^2)$, where n is the size (number of nodes) of the TP. A notable exception is Chen and Chan's extension of ICs to FSBT constraints [44] that makes the matching problem more complex. Thus, their algorithms range in complexity, with respect to the combination of ICs that are processed, from $O(n^2)$ (F and FS ICs) to $O(n^3 \cdot s + n^2 \cdot s^2)$ (FSBT ICs), where s is the size of the set of element types in ICs.

On the other hand, in papers about holistic approaches, complexity is often disregarded in favor of the class of TP a given algorithm is optimal for. The class of TP may be [75]:

- *AD:* a TP containing only AD edges, which may begin with a PC edge;
- *PC:* a TP containing only PC edges, which may begin with an AD edge;
- *one-branching node TP:* a TP containing at most one branching node;
- *AD in branches TP:* a TP containing only AD edges after the first branching node;
- *PC before AD TP:* a TP where an AD edge is never found before a PC edge.

Holistic approaches indeed all have the same worst-case time complexity. It is linear in the sum of all sizes of the input lists they process *and* the output list. However, since these approaches match twigs, they enumerate all root-to-leaf path matches. Thus, their complexity is actually exponential in n, i.e., it is $O(d^n)$, where d is the size (number of nodes) of the input data tree [69]. Only Twig^{2}Stack has a lower complexity, in $O(d \cdot b)$, where $b = max(b_1, b_2)$ with b_1 being the maximum number of query nodes with the same label and b_2 the maximum total number of children of query nodes with the same labels ($b \leq n$) [69]; as well as TreeMatch [36], which has been experimentally shown to outperform Twig^{2}Stack.

Structural summary-based approaches (Section 4.2.3) have been experimentally shown to perform matching faster than previous holistic algorithms such as TwigStack and TJFast. However, they have neither been compared to one another, nor to TreeMatch. Moreover, their time complexity is expressed in terms of DataGuide degree and specific features such as the number of versions in TwigVersion [73], so it is not easy to directly compare it to other holistic algorithms'. For instance, TwigVersion has a worst case time complexity of $O(l_v \cdot n_v + l_p \cdot e)$, where l_v is the depth of version tree (annotated DataGuide) v, n_v is the total number of versions on nodes of v whose tags appear in the leaf nodes of TP p, l_p the depth of TP p, and e is the size of the edge set containing all edges that are on paths from nodes of v whose tags appear in the leaf nodes of p to the root node of v [73].

Finally, the complexity of tree homeomorphism matching is proved to be $O(n \cdot d \cdot l_p)$ [74].

4.4.2 Space complexity

Space complexity is intensively addressed in the literature regarding holistic approaches, which can produce many intermediate results whose volume must be minimized, so that algorithms can run in memory with as low response time as possible. On the other hand, space complexity is not considered an issue in minimization processes, which prune nodes in PTs that are presumably small enough to fit into any main memory.

Regarding holistic approaches, the worst-case space complexity of TwigStack is $min(n \cdot l_t, s)$ [59], where n is defined as in Section 4.4.1, l_t is the depth of data tree t and s is the sum of sizes of the n input lists in the computing phase (Section 4.2.2). Subsequent holistic algorithms are more time-efficient than Twigstack because they are more space-efficient. TwigStackList and iTwigJoin both have a worst-case space complexity of $O(n \cdot l_t)$ [66], [68]. Noticeably, Twig^{2}Stack's space complexity is indeed the same as its time complexity, i.e., $O(d \cdot b)$ (Section 4.4.1). Finally, TJFast and TreeMatch have a worst-case space complexity of $O(l_t^2 \cdot bf + l_t \cdot f)$, where bf is the maximal branching factor of the input data tree and f the number of leaf nodes in the TP [36], [61].

Structural summary approaches' space complexity, as is the case for time complexity, are difficult to compare because of their specifics. For example, the worst case space complexity of TwigVersion is $O(l_v \cdot n_v)$, where l_v and n_v are defined as in Section 4.4.1

TABLE 2
Synthesis of tree pattern matching approaches

Algorithm	TP used	Time complexity	Space complexity	Features
CIM [16]	With output node	$O(n^4)$	N/A	Prunes nodes
Coverage [49] **Simulation** [43]	With output node	$O(n^2)$	N/A	Prune subtrees
Recursive pruning [51] [20]	Unspecified Limited branched TP	$O(n^2)$	N/A	Prune subtrees
ACIM [16]	With output node	$O(n^6)$	N/A	FS ICs
CDM [16]	With output node	$O(n^2)$	N/A	FS ICs
Chase + coverage [49] **Chase + simulation** [43]	With output node	$O(n^2)$	N/A	FS ICs
FSBT [44]	With output node	From $O(n^2)$ to $O(n^3 \cdot s + n^2 \cdot s^2)$	N/A	FSBT ICs
TwigStack [59]	Twig pattern	$O(d^n)$	$min(n \cdot l_t, s)$	Region enc.
TwigStackList [66] **OrderedTJ** [67] **iTwigJoin** [68]	Twig pattern Ordered twig Twig pattern	$O(d^n)$	$O(n \cdot l_t)$	Region enc. *Tag+level, Prefix*
Twig^{2}Stack [69]	GTP	$O(d \cdot b)$	$O(d \cdot b)$	Region enc.
TJFast [61]	Twig pattern	$O(d^n)$	$O(l_t^2 \cdot bf + l_t \cdot f)$	Dewey ID
TreeMatch [36]	Extended TP	$O(d \cdot b)$	$O(l_t^2 \cdot bf + l_t \cdot f)$	Dewey ID
TwigX-Guide [72]	Twig pattern	Unspecified	Unspecified	DataGuide + Region enc.
S^3 [34]	TP with logical operators	Unspecified	Unspecified	DataGuide + Dewey ID
TwigVersion [73]	Twig with node predicates	$O(l_v \cdot n_v + l_p \cdot e)$	$O(l_v \cdot n_v)$	DataGuide + Dewey+Version
Tree homeomorphism [74]	Unranked TP	$O(n \cdot d \cdot l_p)$	$O(l_t \cdot log(bf))$	Left-to-right post-order

[73]. Eventually, tree homeomorphism matching algorithms have a space complexity of $O(l_t \cdot log(bf))$ [74].

4.4.3 Synthesis

Table 2 recapitulates the characteristics of all matching optimization algorithms surveyed in this paper. In addition to the complexity comparison criteria discussed above, we also indicate in Table 2 the type of TP handled by each algorithm, in reference to Section 3, as well as each algorithm's distinctive features. The first third of Table 2 is dedicated to minimization approaches, the second to holistic approaches, and the third to tree homeomorphism matching.

In the light of this synthesis, we can notice again that TP minimization and holistic approaches have developed along separate roads. Papers related to one approach seldom refer to the other. Indeed, these two families of approaches cannot be compared, e.g., in terms of raw complexity, since TP minimization operates on TPs only, while holistic approaches optimize the actual matching of a TP against a data tree. TP minimization is actually implicitly considered as preprocessing TPs before matching [76]. We nonetheless find it surprising that nobody has ever combined TP minimization to holistic matching in a single framework to benefit from optimization of both data tree access and TP size.

Eventually, Table 2 clearly outlines the history and outcome of the families of algorithms we survey. With respect to PT minimization algorithms, unless elaborated ICs (i.e., FSBT ICs) are needed, the choice should clearly fall on an approach whose time complexity is $O(n^2)$, such as the coverage-based approaches [43], [49]. With respect to holistic matching, the raw efficiency of algorithms is not always easy to compare, especially between the latest descendants of Twigstack and structural summary-based approaches such as TwigVersion [73] or S^3 [34], for both complexity and experimental studies remain partial as of today. The types of TPs (Section 4.4.1) for which an approach is optimal thus remains a primary criterion. However, we agree with Grimsmo and Bjørklund in stating that a global holistic approach, i.e., an approach that encompasses all kinds of TPs, is most desirable [19]. In this respect, TreeMatch [36] appears as the most comprehensive solution as of today.

5 TREE PATTERN USAGES

BESIDE expressing and optimizing queries over tree-structured documents, TPs have also been

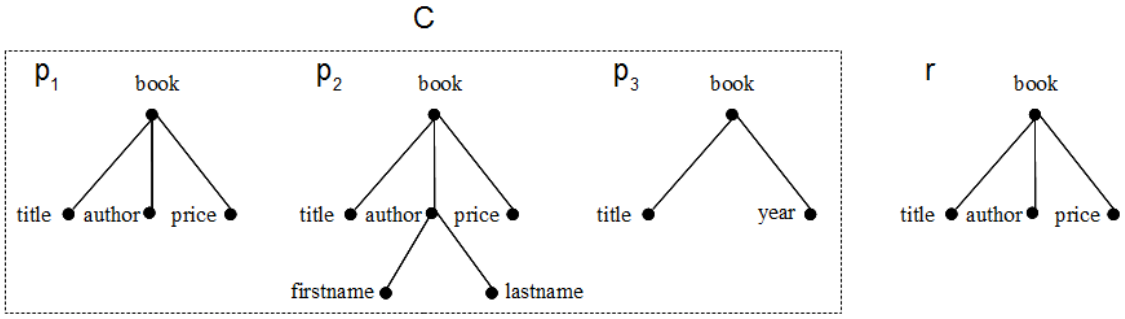

Fig. 15. Sample tree pattern collection and rooted subtree

exploited for various purposes ranging from system optimization (e.g., query caching [77], [78], addressing and routing over a peer-to-peer network [79]) to high-level database operations (e.g., schema construction [80], active XML query satisfiability and relevance [81], [82]) and knowledge discovery (e.g., discovering user communities [83]).

In this section, we investigate the most prominent of TP usages we found in the literature, which we classify by the means used to achieve the goals we have listed above (e.g., routing and query satisfiability), i.e., TP mining (Section 5.1), TP rewriting (Section 5.2) and extensions to matching (Section 5.3).

5.1 Tree pattern mining

TP mining actually summarizes into discovering frequent subtrees in a collection of TPs. It is used, for instance, to cache the results of frequent patterns, which significantly improves query response time [77], produce data warehouse schemas of integrated XML documents from historical user queries [80], or help in website management by mining data streams [84].

5.1.1 Problem formulation

Let $C = \{p_1, p_2, ..., p_n\}$ be a collection of n TPs and $minsup \in \ ]0, 1]$ a number called minimum support. The support of any rooted subtree (denoted RST) r is $sup(r) = freq(r)/n$, where $freq(r)$ is the total occurrence of r in C. Then, the problem of mining frequent RSTs from C may be defined as finding the set $F = \{r_1, r_2, ..., r_m\}$ of m RSTs such that $\forall i \in [1, m], sup(r_i) \geq minsup$ [77].

For example, let us consider TP collection C and RST r from Figure 15. Since $r \subseteq p_1$ and $r \subseteq p_2$, $freq(r) = 2$ and $sup(r) = \frac{2}{3}$. If $minsup = \frac{1}{2}$, then r is considered frequent. Note that searching frequent RSTs relies on testing containment (Section 4.1.1) against TPs of C.

5.1.2 Frequent subtree mining algorithms

One of the first frequent RST mining algorithm, XQPMiner [85], operates like the famous frequent itemset mining algorithm Apriori [86]. XQPMiner initializes by enumerating all frequent 1-edge RSTs. Then, each further step i is subdivided in two substeps:

1) candidate i-edge RSTs are built from $(i-1)$-edge RSTs and filtered with respect to the minimum support;
2) each remaining i-edge RST is tested for containment in each TP of C to compute its support.

FastXMiner optimizes this process by constructing a global query pattern tree (G-QTP, Section 3.2.1) over C. Then, a tree-encoding scheme is applied on the G-QTP to partition candidate RSTs into equivalence classes. The authors show that only single-branch candidate RSTs need to be matched against the TPs from C, which leads to a large reduction in the number of tree inclusion tests [77].

MineFreq further builds upon this principle by mining frequent RST *sets* [80]. A frequent RST set must satisfy two requirements:

1) *support requirement:* $sup(r_1, r_2, ..., r_n) \geq minsup$;
2) *confidence requirement:* $\forall r_i, \frac{freq(r_1, r_2, ..., r_n)}{freq(r_i)} \geq minconf$, where $minconf$ is a minimum confidence user-specified threshold.

The algorithm again proceeds by level, the n-itemset RST candidates being generated by joining $(n-1)$itemset frequent RSTs. Candidates in which one or more $(n-1)$-subsets are infrequent are pruned, for they cannot be frequent. Remaining candidates are then filtered with respect to support and confidence. Finally, frequent TPs are built by joining all RSTs in each frequent RST set.

Eventually, Stream Tree Miner (STMer) mines frequent labeled ordered subtrees over a tree-structured data stream [84]. Its main contribution lies in candidate subtree generation, which is suitably incremental.

5.2 Tree pattern rewriting

Query rewriting is casually used when views, whether they are materialized or not, are defined over data. Rewriting a query Q that runs against a database D is formulating a query Q' that runs against a view V built from D such that Q and Q' output the same

result. When data are tree-structured, views are predefined TPs. Then, Q' is found among so-called useful embeddings (UEs) of Q in V; the problem being to prune redundant, useless UEs [78]. The heuristics that address this issue [78], [87], [88] rely on containment testing (Section 4.1.1) to output a minimal set of UEs.

Query rewriting is also notably used in Peer Data Management Systems (PDMSs). In a PDMS, each peer is associated with a schema that represents the peer's domain of interest, as well as semantic relationships to neighboring peers. Thus, a query over one peer can obtain relevant data from any reachable peer in the PDMS. Semantic paths are traversed by reformulating queries at a peer into queries on its neighbors, and then to the neighbors' neighbors, recursively. In the Piazza PDMS [79], data is modeled in XML, peer schemas in XML Schema, and queries in a subset of XQuery. Thus, query reformulation optimization strongly relates to TP matching optimization. For instance, one query may follow multiple paths in a PDMS, and thus may induce redundant reformulations, i.e., redundant queries on the peers. Pruning queries help reduce such redundancy. Pruning requires checking query containment (Section 4.1.1) between a previously obtained reformulation and a new one. The reformulating process may also introduce redundant subexpressions in a query, leading to query minimization (Section 4.1).

5.3 Extended matching

We review in this section two ways of "pushing" matching further on. The first one relates to checking the satisfiability of TPs against Active XML (AXML) documents [81], [82]. AXML documents contain both data defined in extension (as in XML documents) and in intention, by means of Web service calls [89]. When a Web service is invoked, its result is inserted into the document. In this context, *a priori* checking whether a query (TP) is satisfiable, i.e., whether there exists any document the TP matches against, helps avoid unnecessary query computations.

Although TP satisfiability is well studied [22], with AXML documents, Web service calls may also return data that contribute to query results, which makes the problem even more complex. Here, some fact is satisfiable for an AXML document and a query if it can be in the query result *in some future state* [82]. As Miklau and Suciu did for containment testing (Section 4.1.1), Ma *et al.* use tree automata to represent both TPs and sets of AXML documents conforming to a given schema. Then, the product tree automaton helps decide whether a TP matches any document. Abiteboul *et al.* further test the relevance of Web service calls, i.e., whether the result can impact query answer [82].

The other example of matching enhancement lies in the context of building semantic communities, i.e., clusters of users with similar interests modeled as TPs. Chand *et al.* propose to replace equivalence by similarity in matching, which could more generally be used to approximate XML queries [83], as TP relaxation (Section 4.3.2). Chand *et al.* formulate the TP similarity problem as follows. Let S be a set of TPs, D a set of data trees, and $p, q \in S$. The similarity of p and q is a function $sim : S^2 \mapsto [0, 1]$ such that $sim(p, q)$ is the probability that p matches the same subset of data from D as q. Note that, depending on the proximity metric, $sim(p, q)$ may be different from $sim(q, p)$.

6 Conclusion

We provide in this paper a comprehensive survey about XML tree patterns, which are nowadays considered crucial in XML querying and its optimization. We first compare TPs from a structural point of view, concluding that the richer a TP is with matching possibilities, the larger the subset of XQuery/XPath it encompasses, and thus the closer to user expectations it is.

Secondly, acknowledging that TP querying, i.e., matching a TP against a data tree, is central in TP usage, we review methods for TP matching optimization. They belong to two main families: TP minimization and holistic matching. We trust we provide a good overview of these approaches' evolution, and highlight the best algorithms in each family as of today. Moreover, we want to emphasize that TP minimization and holistic matching are complementary and should both be used to wholly optimize TP matching.

We eventually illustrate how TPs are actually exploited in several application domains such as system optimization, network routing or knowledge discovery from XML sources. We especially demonstrate the use of frequent TP mining and TP rewriting for various purposes.

Although TP-related research, which has been ongoing for more than a decade, could look mature in the light of this survey, it is perpetually challenged by the ever-growing acceptance and usage of XML. For instance, recent applications require either querying data with a complex or only partially known structure, or integrating heterogeneous XML data sources (e.g., when dealing with streams). The keyword search-based languages that address these problems cannot be expressed with TPs [90]. Thus, TPs must be extended, e.g., by so-called partial tree-pattern queries (PTPQs) that allow the partial specification of a TP and are not restricted by a total order on nodes [90], [91]. In turn, adapted matching procedures must be devised [92], a trend that is likely to perpetuate in the near future.

Moreover, we purposely focus on the core of TP-related topics in this survey (namely, TPs themselves,

matching issues and a couple of applications). There is nonetheless a large number of important topics that we could not address due to space constraints, such as TP indexing, TP-based view selection, TP for probabilistic XML and continuous TP matching over XML streams.

ACKNOWLEDGMENTS

We would like to thank all the anonymous reviewers of this paper for their thoughtful comments, and especially the reviewers of the very first version, who expressed interest in this work and encouraged us to write this widely complemented version.

We would also like to thank Mr. Huayu Wu, from the Institute of InfoComm Research, Singapore, for his valuable correspondence.

REFERENCES

[1] L. Quin, "Extensible Markup Language (XML)," http://www.w3.org/XML/, World Wide Web Consortium (W3C), 2006.

[2] D. Carlisle, P. Ion, and R. Miner, "Mathematical Markup Language (MathML) Version 3.0," http://www.w3.org/TR/MathML/, World Wide Web Consortium (W3C), 2010.

[3] P. Murray-Rust and H. Rzepa, "Chemical Markup Language - CML," http://www.xml-cml.org/, 1995.

[4] R. Lake, D. S. Burggraf, M. Trninic, and L. Rae, *Geography Mark-Up Language: Foundation for the Geo-Web*. Wiley, 2004.

[5] ADL, "SCORM 2004 4th Edition Version 1.1 Overview," http://www.adlnet.gov/Technologies/scorm/, Advanced Distributed Learning (ADL), 2004.

[6] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0," http://www.w3.org/TR/xpath, World Wide Web Consortium (W3C), 1999.

[7] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon, "XQuery 1.0: An XML Query Language," http://www.w3.org/TR/xquery/, World Wide Web Consortium (W3C), 2007.

[8] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson, "TAX: A Tree Algebra for XML." in *8th International Workshop on Database Programming Languages (DBPL 01), Frascati, Italy*, ser. LNCS, vol. 2397. Springer, 2001, pp. 149–164.

[9] A. Trotman, N. Pharo, and M. Lehtonen, "XML-IR Users and Use Cases," in *5th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 06), Dagstuhl Castle, Germany*, ser. LNCS, vol. 4518, 2006, pp. 400–412.

[10] P. Michiels, G. A. Mihaila, and J. Siméon, "Put a Tree Pattern in Your Algebra," in *23rd International Conference on Data Engineering (ICDE 07), Istanbul, Turkey*. IEEE, 2007, pp. 246–255.

[11] A. Deutsch, M. F. Fernández, and D. Suciu, "Storing Semistructured Data with STORED," in *ACM SIGMOD International Conference on Management of Data (SIGMOD 99), Philadelphia, PA, USA*. ACM Press, 1999, pp. 431–442.

[12] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," in *25th International Conference on Very Large Data Bases (VLDB 99), Edinburgh, Scotland, UK*. Morgan Kaufmann, 1999, pp. 302–314.

[13] S. Paparizos and H. V. Jagadish, "Pattern Tree Algebras: Sets or Sequences?" in *31st International Conference on Very Large Data Bases (VLDB 05), Trondheim, Norway*. ACM, 2005, pp. 349–360.

[14] S. Paparizos and H. Jagadish, "The Importance of Algebra for XML Query Processing," in *2nd International Workshop on Database Technologies for Handling XML Information on the Web (DataX 06), Munich, Germany*, ser. Lecture Notes in Computer Science, vol. 4254. Springer, 2006, pp. 126–135.

[15] D. D. Chamberlin and R. F. Boyce, "SEQUEL: A Structured English Query Language," in *In Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control (SIGMOD Workshop, Vol. 1 1974), Ann Arbor, Michigan, USA*, 1974, pp. 249–264.

[16] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava, "Minimization of Tree Pattern Queries," in *ACM SIGMOD 20th International Conference on Management of Data (SIGMOD 01), Santa Barbara, California, USA*, 2001, pp. 497–508.

[17] G. Gou and R. Chirkova, "Efficiently Querying Large XML Data Repositories: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 10, pp. 1381–1403, 2007.

[18] L. V. S. Lakshmanan, "XML Tree Pattern, XML Twig Query," in *Encyclopedia of Database Systems*. Springer US, 2009, pp. 3637–3640.

[19] N. Grimsmo and T. A. Bjørklund, "Towards unifying advances in twig join algorithms," in *21st Australasian Database Conference (ADC 10), Brisbane, Australia*, ser. CRPIT, vol. 104. Australian Computer Society, 2010, pp. 57–66.

[20] S. Flesca, F. Furfaro, and E. Masciari, "On the minimization of Xpath queries," in *29th International Conference on Very Large Data Bases (VLDB 03), Berlin, Germany*, 2003, pp. 153–164.

[21] Z. Chen, H. V. Jagadish, L. V. S. Lakshmanan, and S. Paparizos, "From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery," in *29th International Conference on Very Large Data Bases (VLDB 03), Berlin, Germany*, 2003, pp. 237–248.

[22] L. V. S. Lakshmanan, G. Ramesh, H. Wang, and Z. J. Zhao, "On Testing Satisfiability of Tree Pattern Queries," in *30th International Conference on Very Large Data Bases (VLDB 04), Toronto, Canada*. Morgan Kaufmann, 2004, pp. 120–131.

[23] J. Wang, J. X. Yu, and C. Liu, "Independence of Containing Patterns Property and Its Application in Tree Pattern Query Rewriting Using Views," *World Wide Web*, vol. 12, no. 1, pp. 87–105, 2009.

[24] D. Beech, A. Malhotra, and M. Rys, "A formal data model and algebra for XML," W3C XML Query Working Group Note, Tech. Rep., 1999.

[25] C. Beeri and Y. Tzaban, "SAL: An Algebra for Semistructured Data and XML," in *WebDB (Informal Proceedings), Philadelphia, USA*, 1999, pp. 37–42.

[26] D. D. Chamberlin, J. Robie, and D. Florescu, "Quilt: An XML Query Language for Heterogeneous Data Sources," in *3rd International Workshop on The World Wide Web and Databases (WebDB 00), Dallas, Texas, USA, May 18-19*, ser. Lecture Notes in Computer Science, vol. 1997. Springer, 2000, pp. 1–25.

[27] A. Deutsch, M. F. Fernández, D. Florescu, A. Levy, and D. Suciu, "XML-QL: A Query Language for XML," http://www.w3.org/TR/NOTE-xml-ql/, World Wide Web Consortium (W3C), 1998.

[28] H. Ishikawa, K. Kubota, Y. Kanemasa, and Y. Noguchi, "The Design of a Query Language for XML Data," in *10th International DEXA Workshop on Database and Expert Systems Applications, Florence, Italy*, 1999.

[29] G. Mecca, P. Merialdo, and P. Atzeni, "Do we really need a new query language for XML?" in *1st W3C Query Languages Workshop (QL 98), Boston, USA*, 1998.

[30] N. Shinagawa, H. Kitagawa, and Y. Ishikawa, "X^2QL: An eXtensible XML Query Language Supporting User-Defined Foreign Functions," in *2000 ADBIS-DASFAA Symposium on Advances in Databases and Information Systems (ADBIS-DASFAA 00), Prague, Czech Republic*, 2000, pp. 251–264.

[31] S. Paparizos, Y. Wu, L. V. S. Lakshmanan, and H. V. Jagadish, "Tree Logical Classes for Efficient Evaluation of XQuery," in *SIGMOD 23rd International Conference on Management of Data (SIGMOD 04), Paris, France*. ACM, 2004, pp. 71–82.

[32] Y. Chen, "Discovering Ordered tree patterns from XML queries," in *8th Pacific-Asia Conference In Advances in Knowledge Discovery and Data Mining (PAKDD 04), Sydney, Australia*, 2004, pp. 559–563.

[33] B. Kimelfeld and Y. Sagiv, "Twig Patterns: From XML Trees to Graphs," in *9th International Workshop on the Web and Databases (WebDB 06), Chicago, Illinois, USA*, 2006.

[34] S. K. Izadi, T. Härder, and M. S. Haghjoo, "S^3: Evaluation of Tree-Pattern Queries Supported by Structural Summaries," *Data & Knowledge Engineering*, vol. 68, no. 1, pp. 126–145, 2009.

[35] G. Miklau and D. Suciu, "Containment and Equivalence for an XPath Fragment," in *ACM SIGACT-SIGMOD-SIGART 21st Symposium on Principles of Database Systems (PODS 02), Madison, USA*, 2002, pp. 65–76.

[36] J. Lu, T. W. Ling, Z. Bao, and C. Wang, "Extended XML Tree Pattern Matching: Theories and Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 3, March 2011.

[37] S. Amer-Yahia, S. Cho, and D. Srivastava, "Tree Pattern Relaxation," in *8th International Conference on Extending Database Technology (EDBT 02), Prague, Czech Republic*, ser. LNCS, vol. 2287, 2002, pp. 496–513.

[38] S. Paparizos, S. Al-Khalifa, A. Chapman, H. V. Jagadish, L. V. S. Lakshmanan, A. Nierman, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu, "TIMBER: A Native System for Querying XML." in *ACM SIGMOD 22th International Conference on Management of Data (SIGMOD 03), San Diego, USA*. ACM, 2003, p. 672.

[39] H. Katz, "XQEngine at SourceForge," http://xqengine.sourceforge.net/, Fatdog Software Inc., 2005.

[40] J. Hidders, "Satisfiability of XPath Expressions," in *9th International Workshop On Database Programming Languages (DBPL 03), Potsdam, Germany*, ser. Lecture Notes in Computer Science, vol. 2921. Springer, 2004, pp. 21–36.

[41] C. David, "Complexity of Data Tree Patterns over XML Documents," in *33rd International Symposium on Mathematical Foundations of Computer Science 2008 (MFCS 08), Torun, Poland*, ser. Lecture Notes in Computer Science, vol. 5162. Springer, 2008, pp. 278–289.

[42] M. Benedikt, W. Fan, and F. Geerts, "XPath satisfiability in the presence of DTDs," *Journal of the ACM*, vol. 55, no. 2, 2008.

[43] P. Ramanan, "Efficient algorithms for minimizing tree pattern queries," in *2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 02), Madison, Wisconsin, USA*. ACM, 2002, pp. 299–309.

[44] D. Chen and C. Y. Chan, "Minimization of tree pattern queries with constraints," in *2008 ACM SIGMOD International Conference on Management of Data (SIGMOD 08), Vancouver, BC, Canada*. ACM, 2008, pp. 609–622.

[45] A. K. Chandra and P. M. Merlin, "Optimal Implementation of Conjunctive Queries in Relational Data Bases," in *9th Annual ACM Symposium on Theory of Computing (STOC 77), Boulder, USA*, 1977, pp. 77–90.

[46] F. Neven and T. Schwentick, "XPath Containment in the Presence of Disjunction, DTDs, and Variables," in *9th International Conference on Database Theory (ICDT 03), Siena, Italy*, ser. Lecture Notes in Computer Science, vol. 2572. Springer, 2003, pp. 312–326.

[47] P. T. Wood, "Containment for XPath Fragments under DTD Constraints," in *9th International Conference on Database Theory (ICDT 03), Siena, Italy*, ser. Lecture Notes in Computer Science, vol. 2572. Springer, 2003, pp. 297–311.

[48] Y. Chen and D. Che, "Efficient Processing of XML Tree Pattern Queries," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 10, no. 5, pp. 738–743, 2006.

[49] D. Che and Y. Liu, "Efficient Minimization of XML Tree Pattern Queries," in *1st International Conference on Next Generation Web Services Practices (NWeSP 05), Seoul, Korea*, 2005.

[50] S. Abiteboul and V. Vianu, "Queries and computation on the web," *Theoretical Computer Science*, vol. 239, no. 2, pp. 231–255, 2000.

[51] C. Y. Chan, W. Fan, P. Felber, M. N. Garofalakis, and R. Rastogi, "Tree Pattern Aggregation for Scalable XML Data Dissemination," in *28th International Conference on Very Large Data Bases (VLDB 02), Hong Kong, China*. Morgan Kaufmann, 2002, pp. 826–837.

[52] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.

[53] Y. Chen and D. Che, "Minimization of XML Tree Pattern Queries in the Presence of Integrity Constraints," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 10, no. 5, pp. 744–751, 2006.

[54] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," in *18th International Conference on Data Engineering (ICDE 02), San Jose, CA, USA*. IEEE Computer Society, 2002, p. 141.

[55] J. Lu, "Benchmarking Holistic Approaches to XML Tree Pattern Query Processing - (Extended Abstract of Invited Talk)," in *15th International Conference ON Database Systems for Advanced Applications (DASFAA 10), International Workshops: GDM, BenchmarX, MCIS, SNSMW, DIEW, UDM, Tsukuba, Japan*, ser. Lecture Notes in Computer Science, vol. 6193. Springer, 2010, pp. 170–178.

[56] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," in *27th International Conference on Very Large Data Bases (VLDB 01), Roma, Italy*, 2001, pp. 361–370.

[57] X. Wu, M.-L. Lee, and W. Hsu, "A Prime Number Labeling Scheme for Dynamic Ordered XML Trees," in *20th International Conference on Data Engineering (ICDE 04), Boston, MA, USA*. IEEE Computer Society, 2004, pp. 66–78.

[58] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan, "On the Integration of Structure Indexes and Inverted Lists," in *2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 04), Paris, France*. ACM, 2004, pp. 779–790.

[59] N. Bruno, N. Koudas, and D. Srivastava, "Holistic twig joins: optimal XML pattern matching," in *2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 02), Madison, USA*. ACM, 2002, pp. 310–321.

[60] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang, "Storing and querying ordered xml using a relational database system," in *2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 02), Madison, Wisconsin, USA*. ACM, 2002, pp. 204–215.

[61] J. Lu, T. W. Ling, C. Y. Chan, and T. Chen, "From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching," in *31st International Conference on Very Large Data Bases (VLDB 05), Trondheim, Norway*. ACM, 2005, pp. 193–204.

[62] P. E. O'Neil, E. J. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATHs: Insert-Friendly XML Node Labels," in *2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 04), Paris, France*. ACM, 2004, pp. 903–908.

[63] L. Xu, T. W. Ling, H. Wu, and Z. Bao, "DDE: from Dewey to a fully dynamic XML labeling scheme," in *2009 ACM SIGMOD International Conference on Management of Data (SIGMOD 09), Providence, USA*. ACM, 2009, pp. 719–730.

[64] J. Lu, "Efficient Processing of XML Twig Pattern Matching," Ph.D. dissertation, National University of Singapore, 2006.

[65] A. Barta, M. P. Consens, and A. O. Mendelzon, "Benefits of Path Summaries in an XML Query Optimizer Supporting Multiple Access Methods," in *31st International Conference on Very Large Data Bases (VLDB 05), Trondheim, Norway*. ACM, 2005, pp. 133–144.

[66] J. Lu, T. Chen, and T. W. Ling, "Efficient processing of XML twig patterns with parent child edges: a look-ahead approach," in *2004 ACM CIKM International Conference on Information and Knowledge Management (CIKM 04), Washington, USA*. ACM, 2004, pp. 533–542.

[67] J. Lu, T. W. Ling, T. Yu, C. Li, and W. Ni, "Efficient Processing of Ordered XML Twig Pattern," in *16th International on Database and Expert Systems Applications (DEXA 05), Copenhagen, Denmark*, ser. Lecture Notes in Computer Science, vol. 3588. Springer, 2005, pp. 300–309.

[68] T. Chen, J. Lu, and T. W. Ling, "On Boosting Holism in XML Twig Pattern Matching using Structural Indexing Techniques," in *ACM SIGMOD 24th International Conference on Management of Data (SIGMOD 05), Baltimore, Maryland, USA*. ACM, 2005, pp. 455–466.

[69] S. Chen, H.-G. Li, J. Tatemura, W.-P. Hsiung, D. Agrawal, and K. S. Candan, "Twig^{2}stack: Bottom-up processing of generalized-tree-pattern queries over xml documents," in *32nd International Conference on Very Large Data Bases (VLDB 06), Seoul, Korea*.

[70] R. Baca, M. Krátký, and V. Snásel, "On the efficient search of an XML twig query in large DataGuide trees," in *12th International Database Engineering and Applications Symposium (IDEAS 08), Coimbra, Portugal*, ser. ACM International Conference Proceeding Series, vol. 299. ACM, 2008, pp. 149–158.

[71] R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases,"

in *23rd International Conference on Very Large Data Bases, Athens, Greece*, 1997, pp. 436–445.

[72] S.-C. Haw and C.-S. Lee, "TwigX-Guide: An Efficient Twig Pattern Matching System Extending DataGuide Indexing and Region Encoding Labeling," *Journal of Information Science and Engineering*, vol. 25, no. 2, pp. 603–617, 2009.

[73] X. Wu and G. Liu, "XML twig pattern matching using version tree," *Data & Knowledge Engineering*, vol. 64, no. 3, pp. 580–599, 2008.

[74] M. Götz, C. Koch, and W. Martens, "Efficient algorithms for descendant-only tree pattern queries," *Information Systems*, vol. 34, no. 7, pp. 602–623, 2009.

[75] R. Baca, "Path-based Approaches to the Twig Pattern Query Searching," Ph.D. dissertation, VSB-Technical University of Ostrava, Czech Republic, 2008.

[76] J. Yao and M. Z. II, "A Fast Tree Pattern Matching Algorithm for XML Query," in *2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI 04), 20-24 September 2004, Beijing, China*. IEEE Computer Society, 2004, pp. 235–241.

[77] L. H. Yang, M.-L. Lee, and W. Hsu, "Efficient Mining of XML Query Patterns for Caching," in *29th International Conference on Very Large Data Bases (VLDB 03), Berlin, Germany*, 2003, pp. 69–80.

[78] J. Wang, K. Wang, and J. Li, "Finding Irredundant Contained Rewritings of Tree Pattern Queries Using Views," in *Advances in Data and Web Management, Joint International Conferences (APWeb/WAIM 09), Suzhou, China*, 2009, pp. 113–125.

[79] I. Tatarinov and A. Y. Halevy, "Efficient Query Reformulation in Peer-Data Management Systems," in *2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 04), Paris, France*. ACM, 2004, pp. 539–550.

[80] J. Zhang, T. W. Ling, R. M. Bruckner, and A. M. Tjoa, "Building XML Data Warehouse Based on Frequent Patterns in User Queries," in *5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 03), Prague, Czech Republic*, ser. Lecture Notes in Computer Science, vol. 2737. Springer, 2003, pp. 99–108.

[81] H.-T. Ma, Z.-X. Hao, and Y. Zhu, "Checking Satisfiability of Tree Pattern Queries for Active XML Documents," *INFOCOMP Journal of Computer Science*, vol. 7, no. 1, pp. 11–18, 2008.

[82] S. Abiteboul, P. Bourhis, and B. Marinoiu, "Satisfiability and relevance for queries over active documents," in *28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 09), Providence, USA*. ACM, 2009, pp. 87–96.

[83] R. Chand, P. Felber, and M. N. Garofalakis, "Tree-pattern similarity estimation for scalable content-based routing," in *23rd International Conference on Data Engineering (ICDE 07), Istanbul, Turkey*. IEEE, 2007, pp. 1016–1025.

[84] M. C.-E. Hsieh, Y.-H. Wu, and A. L. P. Chen, "Discovering Frequent Tree Patterns over Data Streams," in *6th SIAM International Conference on Data Mining (SDM 06), Bethesda, MD, USA*. SIAM, 2006.

[85] L. H. Yang, M.-L. Lee, W. Hsu, and S. Acharya, "Mining Frequent Query Patterns from XML Queries," in *8th International Conference on Database Systems for Advanced Applications (DASFAA 03), Kyoto, Japan*. IEEE Computer Society, 2003, pp. 355–362.

[86] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *20th International Conference on Very Large Data Bases (VLDB 94), Santiago de Chile, Chile*, 1994, pp. 487–499.

[87] A. Balmin, F. Özcan, K. S. Beyer, R. Cochrane, and H. Pirahesh, "A Framework for Using Materialized XPath Views in XML Query Processing," in *30th International Conference on Very Large Data Bases (VLDB 04), Toronto, Canada*. Morgan Kaufmann, 2004, pp. 60–71.

[88] L. V. S. Lakshmanan, W. H. Wang, and Z. J. Zhao, "Answering tree pattern queries using views," in *32nd International Conference on Very Large Data Bases (VLDB 06), Seoul, Korea*, 2006, pp. 571–582.

[89] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber, "Active xml: Peer-to-peer data and web services integration," in *28th International Conference on Very Large Data Bases (VLDB 02), Hong Kong, China*. Morgan Kaufmann, 2002, pp. 1087–1090.

[90] D. Theodoratos and X. Wu, "Eager Evaluation of Partial Tree-Pattern Queries on XML Streams," in *14th International Conference on Database Systems for Advanced Applications (DASFAA 09), Brisbane, Australia*, ser. Lecture Notes in Computer Science, vol. 5463. Springer, 2009, pp. 241–246.

[91] P. Placek, D. Theodoratos, S. Souldatos, T. Dalamagas, and T. K. Sellis, "A heuristic approach for checking containment of generalized tree-pattern queries," in *17th ACM Conference on Information and Knowledge Management (CIKM 08), Napa Valley, California, USA*. ACM, 2008, pp. 551–560.

[92] X. Wu, D. Theodoratos, S. Souldatos, T. Dalamagas, and T. K. Sellis, "Efficient Evaluation of Generalized Tree-Pattern Queries with Same-Path Constraints," in *21st International Conference Scientific and Statistical Database Management (SSDBM 09), New Orleans, LA, USA*, ser. Lecture Notes in Computer Science, vol. 5566. Springer, 2009, pp. 361–379.

Marouane Hachicha received his M.Sc. in computer science from the University of Lyon 2, France in 2007. He has been a Ph.D. student at the University of Lyon 2 since then. He spent six months as a foreign researcher in Italy in 2009. His research interests lie in XML data warehousing and XOLAP.

Jérôme Darmont received his Ph.D. in computer science from the University of Clermont-Ferrand II, France in 1999. He joined the University of Lyon 2, France in 1999 as an associate professor, and became full professor in 2008. He was head of the Decision Support Databases research group within the ERIC laboratory from 2000 to 2008, director of the Computer Science and Statistics Department of the Faculty of Economics and Management from 2003 to 2010, and has been in charge of the Complex Data Warehousing and OLAP research axis at ERIC since 2010. His current research interests mainly relate to handling so-called complex data in data warehouses (XML warehousing, performance optimization, auto-administration, benchmarking...), but also include data quality and security, cloud business intelligence, as well as medical or health-related applications.