Knowl Inf Syst (2016) 46:185–212
DOI 10.1007/s10115-015-0820-0

REGULAR PAPER

Clustering XML documents by patterns

Maciej Piernik · Dariusz Brzezinski · Tadeusz Morzy

Received: 8 August 2013 / Revised: 22 October 2014 / Accepted: 8 January 2015 /
Published online: 23 January 2015

Abstract Now that the use of XML is prevalent, methods for mining semi-structured documents have become even more important. In particular, one of the areas that could greatly benefit from in-depth analysis of XML's semi-structured nature is cluster analysis. Most of the XML clustering approaches developed so far employ pairwise similarity measures. In this paper, we study clustering algorithms, which use patterns to cluster documents without the need for pairwise comparisons. We investigate the shortcomings of existing approaches and establish a new pattern-based clustering framework called XPattern, which tries to address these shortcomings. The proposed framework consists of four steps: choosing a pattern definition, pattern mining, pattern clustering, and document assignment. The framework's distinguishing feature is the combination of pattern clustering and document-cluster assignment, which allows to group documents according to their characteristic features rather than their direct similarity. We experimentally evaluate the proposed approach by implementing an algorithm called PathXP, which mines maximal frequent paths and groups them into profiles. PathXP was found to match, in terms of accuracy, other XML clustering approaches, while requiring less parametrization and providing easily interpretable cluster representatives. Additionally, the results of an in-depth experimental study lead to general suggestions concerning pattern-based XML clustering.

Keywords XML · Semi-structured document analysis · Pattern-based clustering · Pattern definition

1 Introduction

XML became an official W3C Recommendation [5] on February 10, 1998, and since then it has become one of the most popular ways of representing digital information. Countless

M. Piernik (✉) · D. Brzezinski · T. Morzy
Institute of Computing Science, Poznan University of Technology,
ul. Piotrowo 2, 60-965 Poznan, Poland
e-mail: maciej.piernik@cs.put.poznan.pl

applications of XML have inspired the development of hundreds of domain-specific languages, including information technology: *SOAP*—message exchange protocol [41], healthcare: *CDA*—part of the *HL7* standard [13], bioinformatics: *PDBML*—Protein Data Bank XML [42], and mathematics: *MathML*—mathematical notation language [40].

Such a rapid expansion of this standard led to the point, where huge amounts of XML are being generated every day. These data constitute a potentially important source of business and scientific knowledge, which due to its size requires automated processing. In order to automatically extract knowledge from such amounts of data, XML mining methods need to be employed [15]. One of the most important XML mining tasks is XML clustering, which partitions a dataset into groups of presumably similar documents. The key to successful clustering lies in the definition of a similarity measure.

In the context of XML, there are three ways of measuring similarity between documents: omitting the structure (metadata, hierarchy) of documents and treating them as ordinary text documents [23,35], omitting the content of documents and relying solely on structure [6,24,26,28], or considering both structure and content [37,44,45]. However, it is believed that structural information contained in XML documents cannot be ignored and algorithms dedicated to processing text documents are inappropriate for XML document clustering [10]. In this paper, we will focus on structural approaches.

1.1 Shortcomings of existing approaches

Given a dataset $\mathcal{D}$ of n objects, the purpose of clustering is to divide $\mathcal{D}$ into k groups of objects (clusters), such that objects within clusters are more similar to one another than to objects from different clusters. The most common categorization of clustering methods divides them into partitional and hierarchical approaches. Partitional clustering starts with an initial partitioning and iteratively improves it until reaching an algorithm-specific stop condition. Hierarchical clustering methods iteratively split/merge clusters and produce a hierarchy, called a dendrogram, which reflects the order in which clusters were split/merged. The most common hierarchical method is the agglomerative hierarchical clustering algorithm (AHC), which merges two most similar clusters at each iteration. The three most popular merging strategies are as follows: single link, complete link, and average link, in which the distance between clusters is computed as the closest, furthest, and average distance between objects, respectively.

In structural XML clustering, both partitional and hierarchical methods are used. The characteristic feature of structural clustering algorithms lies in the definition of the similarity measure used to group documents. One of the basic approaches to calculating document similarity is the tag-only approach, which measures the number of common tags between each pair of documents. However, for documents which differ mostly in structure rather than tag counts, this approach gives a poor estimate of similarity between documents. This shortcoming is illustrated in Example 1. For the ease of presentation, we will discuss examples using the single link agglomerative hierarchical clustering algorithm; however, using complete link, average link, or a partitional method, such as k-means, would yield an identical result in terms of clustering quality.

Example 1 Let us consider a dataset consisting of eight documents, presented in Table 1. Documents d_1, d_2, d_3, d_4 represent summaries of books, while documents d_5, d_6, d_7 and d_8 contain basic information about journal articles. We will expect the analyzed clustering algorithms to distinguish books from journal articles.

In the tag-only approach, each document is represented as a vector containing the number of occurrences of each tag in the document. Table 2 presents the example documents as tag-

Table 1 Example XML documents describing books (d_1, d_2, d_3, d_4) and journal articles (d_5, d_6, d_7, d_8)

d_1	d_2	d_3	d_4
<paper>	<paper>	<paper>	<paper>
<authors>	<authors>	<authors>	<authors>
<author />	<author />	<author />	<author />
</authors>	</authors>	<author />	<author />
<booktitle>	<booktitle>	</authors>	</authors>
<title />	<title />	<booktitle>	<booktitle>
<year />	<year />	<title />	<title />
<volume />	</booktitle>	<year />	<year />
</booktitle>	<note />	<volume />	<number />
</paper>	</paper>	</booktitle>	</booktitle>
		<pages />	</paper>
		</paper>	
d_5	d_6	d_7	d_8
<paper>	<paper>	<paper>	<paper>
<authors>	<authors>	<authors>	<authors>
<author />	<author />	<author />	<author />
</authors>	</authors>	<author />	<author />
<journal>	<journal>	</authors>	</authors>
<title />	<title />	<journal>	<journal>
<year />	<year />	<title />	<title />
<volume />	</journal>	<year />	<year />
</journal>	<note />	<volume />	<number />
</paper>	</paper>	</journal>	</journal>
		<pages />	</paper>
		</paper>	

Table 2 XML documents from Table 1 represented as tag-only vectors

Tag	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8
paper	1	1	1	1	1	1	1	1
authors	1	1	1	1	1	1	1	1
author	1	1	2	2	1	1	2	2
title	1	1	1	1	1	1	1	1
year	1	1	1	1	1	1	1	1
booktitle	1	1	1	1	0	0	0	0
journal	0	0	0	0	1	1	1	1
volume	1	0	1	0	1	0	1	0
pages	0	0	1	0	0	0	1	0
notes	0	1	0	0	0	1	0	0

only vectors. Using euclidean distance as a similarity measure between the documents, we obtain a dendrogram shown in Fig. 1. As a result, we would expect two clusters representing the two types of documents—books and journal articles. However, for $k = 2$, the tag-only approach fails to identify correct clusters, grouping documents d_3 and d_7 into one cluster and documents $d_1, d_2, d_4, d_5, d_6, d_8$ into another.

The problem with the presented approach is that it discards the document hierarchy. For heterogeneous datasets, simple tag counts are sometimes sufficient, but homogeneous datasets, like the one in the discussed example, require more complex approaches. One of the most studied methods which compares entire document structures is the tree-edit distance, defined as the minimal number of atomic operations required to transform one tree into another [30]. Typically, tree-edit distance is associated with three atomic edit operations

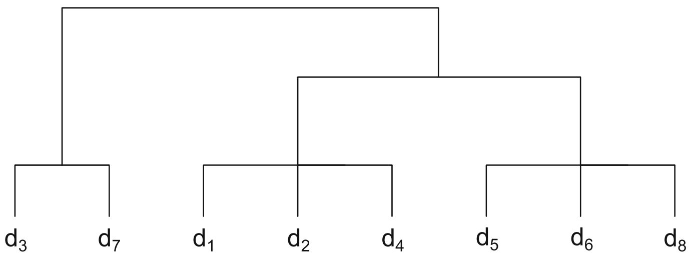


Fig. 1 Dendrogram representing the clustering of tag-only representations from Table 2

Table 3 Distance matrix of the example XML documents from Table 1 created using tree-edit distance

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8
d_1	0							
d_2	2	0						
d_3	2	3	0					
d_4	2	3	2	0				
d_5	1	3	3	3	0			
d_6	3	1	4	4	2	0		
d_7	3	4	1	3	2	3	0	
d_8	3	4	3	1	2	3	2	0

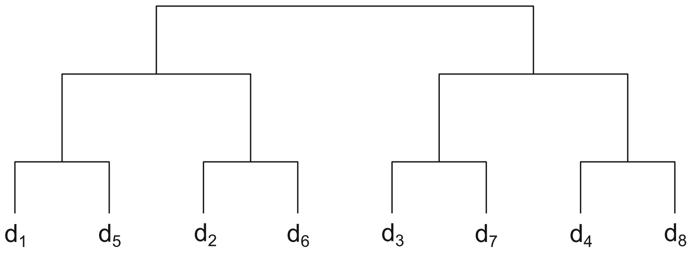


Fig. 2 Dendrogram representing the clustering based on a distance matrix from Table 3

conducted on nodes of a rooted ordered labeled tree: deletion, insertion, and replacement. Let t_1 and t_2 be a pair of rooted ordered labeled trees. A tree-edit sequence is a sequence of tree operations that transform t_1 to t_2. If we assign a cost to every operation, the tree-edit distance between t_1 and t_2 will be the minimum cost of all possible tree-edit sequences that transform t_1 to t_2.

The tree-edit distance measure utilizes the entire structural information encapsulated in XML documents, and in this aspect, it is one of the most complete approaches to computing tree similarity. However, in the context of clustering, this approach can be very sensitive to subtle structural changes, which should be treated as noise. This shortcoming is illustrated in Example 2.

Example 2 Let us once again consider the dataset presented in Table 1. Using a tree-edit distance measure with equally weighted edit operations, we obtain a distance matrix shown in Table 3. A single-link agglomerative hierarchical clustering algorithm applied to this matrix produces a dendrogram presented in Fig. 2. As we can see, the tree-edit distance approach also fails to identify correct clusters, grouping documents d_1, d_2, d_5 and d_6 into one cluster, and documents d_3, d_4, d_7 and d_8 into another.

The reason why the tree-edit distance approach produces incorrect results is because (like most XML clustering algorithms) it uses only *local information*, i.e., information obtained by comparing a pair of documents. Approaches that use only local information do not capture the characteristic features of clusters available in the context of the entire dataset, i.e., *global information*. That is why, in this article, we propose a clustering framework, which uses patterns derived from the entire dataset. Our goal is to introduce a universal way of incorporating global information in structural clustering algorithms. Additionally, we identify five issues, characteristic for pattern-based as well as general clustering approaches, which need to be addressed in order to create a comprehensive XML clustering solution.

The first issue with clustering algorithms is that they usually lack in description. Clusters are often represented by numeric values (e.g., averages and standard deviations), which do not generalize cluster contents in a human-understandable manner. An alternative approach involves describing a cluster by a single object from the cluster. Unfortunately, such a representative often displays individual characteristics instead of modeling a profile that is common for the whole cluster.

A different challenge, characteristic for clustering by patterns, is the choice of a pattern definition. The decision on which definition to choose is crucial to the clustering process as it defines possible similarity measures and the interpretability of the output clusters. Examples of patterns in the domain of XML documents include: frequent subtrees, paths, words or expressions, height or width of the tree, and number of distinct edges. Patterns can also be defined as combinations of these features, which can provide additional flexibility and description, but possibly at the cost of cluster interpretability. This trade-off between description completeness and interpretability makes defining a pattern a difficult task.

Another salient issue for clustering algorithms is parametrization. One of the most important parameters for pattern mining approaches is *minimum support*—a value indicating the minimal percentage of objects containing a feature required to call this feature a pattern. Minimum support highly affects the number of obtained patterns and thus the whole clustering process. If the value is too low, then the number of patterns can be much higher than the number of documents, which can significantly extend clustering time. If the value is too high, there may be very few patterns and each of them will cover too many documents, which may result in poor clustering quality. Therefore, choosing a proper minimum support value is a challenging yet very important task. Furthermore, many algorithms require that the number of result clusters is known a priori. Such a requirement can be unacceptable in many real-life applications and solutions that automatically detect the number of clusters are needed.

The choice of the minimum support value is connected with another problem. Once patterns are chosen, clustering is performed according to their definitions, but there is no guarantee that the patterns will cover all of the documents in the dataset. If the clustering method is a one-pass algorithm, the left out documents have to be processed separately. Leaving them unassigned may be intentional (outlier treatment), but can result in lower clustering quality measured by some evaluation indices.

Lastly, an issue that needs to be concerned is that pattern-based approaches usually rely on a binary similarity measure between documents and patterns—either a document contains a pattern or it does not. However, the number of connections between patterns and documents can be used while assigning documents to clusters, possibly enhancing clustering quality.

In sum, XML document clustering and clustering by patterns pose many problems, which existing algorithms only partially address. In our opinion, the main five challenges in this area are as follows:

C1 Cluster interpretability: describing each cluster in a way that allows for a straightforward interpretation,
C2 Pattern definition: choosing an accurate yet easily interpretable pattern definition,
C3 Parametrization: reducing the number of parameters required by clustering algorithms,
C4 Dataset coverage: deciding how to interpret documents that do not match to any pattern,
C5 Pattern-document similarity: deciding whether to take the number of occurrences of a pattern in a single document into account.

1.2 Our contributions

In this paper, we discuss challenges presented in the previous section and propose a new clustering algorithm called PathXP, which addresses these challenges. PathXP uses groups of frequent paths, called profiles, to cluster XML documents in a divisive manner. The algorithm takes into account not only path-document existence, but also utilizes information about multiple path occurrences and their uniqueness. PathXP follows a more general approach, which we encapsulate in a framework called XPattern.

The main contributions of this paper are as follows:

- We propose an extensible framework for clustering XML documents by patterns, called XPattern.
- We put forward the PathXP algorithm—an instantiation of the XPattern framework that uses maximal frequent paths as patterns.
- We analyze the general components of pattern-based XML clustering algorithms and show their impact on clustering quality.
- We propose a heuristic for estimating the number of clusters k for pattern-based clustering algorithms.
- We evaluate PathXP and discuss alternative pattern definitions.

The remainder of the paper is organized as follows. Section 2 reviews existing XML clustering methods and compares PathXP with similar approaches. Section 3 discusses the steps of the XPattern framework. An in-depth description of PathXP is given in Sect. 4. Section 5 outlines experiments conducted to evaluate the proposed algorithm and analyzes different pattern definitions. Section 6 draws conclusions and suggests lines of further research.

2 Related work

The most popular criterion used to categorize XML clustering methods is the type of document information used for clustering: content, structure, or both content and structure. As this paper concentrates on clustering using structural patterns, we will focus mainly on structure-based methods. Further categorization of XML clustering algorithms can be done by distinguishing different document representations and similarity measures. In the following sections, we summarize existing XML clustering methods based on the techniques they exploit: schema-based, tag/path similarity, vector-based, entropy, edit distance, and pattern approaches.

2.1 DTD approaches

One of the earliest works concerning XML document clustering is the *XClust* algorithm [24]. XClust is designed to cluster XML schemas represented in the form of Document-Type

Definitions (DTDs). Since DTDs are tree structures that can contain artificial AND-OR nodes and can specify cardinality for regular nodes, the authors propose to simplify these trees prior to clustering. The simplification process is performed by applying a set of transformation rules on the original DTD. After simplification, XClust compares DTD nodes with a similarity measure that takes into account their semantics, immediate descendants, and leaf descendants. The semantic similarity involves using the WordNet thesaurus [16] as well as comparing element cardinality and path similarity. Immediate descendants of compared elements are analyzed using their WordNet distance and node cardinality. Finally, similarity between paths of descendant leafs of the compared nodes is added to the overall measure. After calculating schema distances, cluster analysis is performed in a hierarchical manner. A similar approach was presented in the *XMine* algorithm [28].

2.2 Tag and path similarity approaches

Tag similarity approaches are considered to be the simplest algorithms for clustering XML documents [14]. They are presented as a computationally efficient alternative for more complex structural similarity methods such as tree-edit distance algorithms [30]. Tag-based approaches discard the content of an XML document and treat it as a bag of tags. Based on the frequencies of tags and their intersection between documents, similarity between documents can be computed. However, because tag-based approaches discard not only textual but also most of the structural information, they achieve lower clustering quality compared to other methods [30,38].

A compromise between fast tag-based methods and accurate edit distance approaches is provided by algorithms that use XML paths for clustering. A good example is the *PBClustering* algorithm [26], which describes each XML document as a set of XPaths from root to leaf. Since there can be a large number of root to leaf XPaths, the authors propose to use only frequent paths. A frequent path is defined as a path that occurs in at least $minsup$ percent of documents in a dataset, where $minsup$ is a user-defined minimum support level. After extracting frequent paths, each document is encoded as a vector of bits. Each bit in this vector corresponds to a frequent XPath and is set to 0 by default. If a document contains a frequent XPath, the bit that corresponds to that path is set to 1. Finally, a similarity matrix is created by comparing each pair of bit vectors, and clustering is performed by an agglomerative algorithm. PBClustering can use as little features as tag-based approaches while performing as well as more expensive methods [26]. Other interesting path-based approaches to XML clustering include the use of *s-graphs* [27] and the construction of *SuperTrees* [25].

The PathXP algorithm presented in this paper also uses frequent paths to compare documents and in this way is similar to the PBClustering algorithm. However, unlike PBClustering, PathXP is not limited to the use of only root to leaf paths, but utilizes also shorter paths. Furthermore, our algorithm does not require the specification of minimum support. Finally, while comparing documents, PBClustering takes into account only pattern existence, whereas PathXP utilizes the strength between patterns and documents.

2.3 Vector-based approaches

In the field of text clustering, most methods rely on the Vector Space Model (VSM) [34]. In this model, each document is represented as a vector of terms that appear in a dataset. A typical example of such an approach is the TF-IDF scheme [33], in which each word in a document is described by the ratio of its frequency in the document and its frequency in the whole dataset. Because the classical VSM representation of documents uses only words, it is unsuitable for semi-structured data. That is why new methods have been proposed to represent XML data as vectors in an abstract n-dimensional feature space.

An interesting vector-based approach for clustering XML documents was put forward by Candillier et al. [8]. The authors propose to summarize each document with a set of attribute-value pairs with structural information. These pairs may describe tags, edges, sibling relationships, node positions, and the number of distinct paths starting from the document's root. Such an approach creates as many attributes as many distinct features can be encountered in the training set. Afterward, for a given document, each attribute value is set to the number of occurrences of the feature associated with that attribute (e.g., number of occurrences of a certain tag). After calculating feature vectors for all documents, clustering is performed by a variation of the EM algorithm [8]. Since the number of attributes for a dataset can grow fairly large, the authors propose to perform feature selection for each cluster. To perform this, for each cluster, attributes are weighted by the ratio between local and global standard deviations of attribute values. Afterward, only a user-defined number of highest weighted attributes is kept for further processing.

A different method that partially maps XML data to a vector-based representation was proposed by Hagenbuchner et al. [19]. The authors put forward an algorithm called *SOM-SD*, which extends Kohonen's Self Organizing Map (SOM) [22] and clusters XML documents in an unsupervised fashion. For each document in a dataset, SOM-SD processes document tree nodes one at a time and maps them on the SOM's neuron grid. After presenting all the nodes in a dataset to the grid, similar documents are displayed at the same or close coordinates on the SOM. SOM-SD performed favorably to other structural clustering methods in the INEX 2005 XML Mining Competition [12].

2.4 Entropy and FFT approaches

One of the more efficient XML clustering methods is the entropy-based clustering algorithm [21]. The main idea behind this approach is to compress structural information about documents, compare the lengths of the compressed files, and calculate the normalized compression distance (NCD) between each pair of documents. The calculated distances can be later used by a similarity based clustering algorithm like AHC or k-means. An interesting aspect of this algorithm is that it can work with any type of document representation. The definition of NCD allows such flexibility, as it is based on an approximation of the Kolmogorov complexity, which can be defined for any data object [4]. Although the algorithm can use any type of document representation, clustering results will differ depending on the selected representation. For this reason, the author analyzes four methods of extracting structural information: tags, pairs of tags, paths, and whole document trees. Compared to simple tag and edit distance algorithms, the entropy-based approach requires less time and achieves similar or better clustering accuracy, depending on the selected representation and compression algorithm [21].

The entropy-based method was also compared with another interesting algorithm, one that converts XML documents into time series and compares them using frequency analysis [17]. In this approach, each tag occurrence of an XML element/attribute is considered as an impulse. The combined set of impulses in a document can be treated as a time series. Subsequently, structural similarity between documents is measured by analyzing frequencies obtained from corresponding time series through Fast Fourier Transform (FFT). While this algorithm runs faster than edit distance approaches, clustering experiments conducted on both real and synthetic XML data show that the FFT method yields higher error rates [7].

2.5 Edit distance approaches

Most of the methods discussed so far used DTDs or simple structural information like tags or paths. However, full structural information about XML documents is usually represented in tree form. Methods for comparing whole tree structures are based on edit distance measures, which calculate the number of operations required to transform one tree into another. From the myriad of proposed tree-edit distance algorithms, one of the most sophisticated methods to date is an algorithm put forward by Nierman and Jagadish [30].

In their approach, Nierman and Jagadish propose to represent an XML document as a labeled ordered tree in which inner nodes are tags and leaf nodes are tags or attributes. The algorithm allows three operations on single nodes: relabel, insert leaf, and delete leaf. In contrast to many other tree-edit distance methods, the authors also allow two subtree operations: insert subtree and delete subtree. Subtree insertions and deletions are limited only to subtrees that are already contained in the source/destination tree, that is, if all nodes of the inserted/deleted subtree occur in the source/destination tree, with the same parent–child edge relationships and same sibling order [30]. Edit distance methods provide the most complex document comparison, much needed in XML clustering, but are known to be computationally expensive [27]. Currently, the most efficient tree-edit distance method is the RTED, proposed by Pawlik and Augsten [31].

2.6 Pattern approaches

In the *XProj* framework [1], the authors propose a clustering algorithm that uses frequent substructures (tree edges) as patterns. Initially, the document set is randomly divided into k partitions of equal size. Next, sets of frequent edges are mined from these partitions. These frequent edges (cluster representatives) are later used to calculate similarity among documents. The distance between a document and a set of representatives is defined as the fraction of nodes in the document, which are covered by any representative in the set. According to the computed distances, documents are reassigned to clusters with representatives that are most similar. In subsequent iterations, the algorithm mines new frequent edges and repeats the clustering process until it converges or reaches the maximum number of iterations. XProj was compared with two tree-edit distance methods achieving higher precision [1].

Although PathXP also mines frequent substructures to create cluster profiles, there are several differences compared to the work in [1]. Firstly, our algorithm only requires the expected number of clusters as an input parameter, while XProj expects the number of clusters, mined substructure size, and minimum support. Secondly, XProj is a partitioning algorithm and thus produces flat results. PathXP, on the other hand, returns not only the final groups, but also a hierarchy of clusters. Thirdly, XProj is a randomizable algorithm, and its final results depend on the initial grouping of input documents, while for a given dataset, our algorithm always returns the same result. Finally, the coverage-based similarity criterion used to compare documents with patterns in XProj is based on node co-occurrences, while PathXP takes into account path co-occurrences and their arity.

In [6], we presented an approach that utilized maximal frequent subtrees as patterns, grouped those patterns according to a similarity measure, and assigned XML documents to pattern groups. This paper builds on that work and presents the *XPattern* framework, which generalizes the previously proposed algorithm. Furthermore, in this paper, we consider possible instances of the proposed framework by analyzing: the use of several different pattern definitions, the influence of using all frequent patterns and not only maximal, and the use of an occurrence count for patterns appearing more than once in a document. Moreover, we propose and evaluate the use of *weighted support* as a way of taking into account not only pattern frequency, but also pattern uniqueness. The results of our research are compiled into a new algorithm called *PathXP*, which is presented and discussed in the following sections.

3 XPattern clustering framework

As it was shown in Sect. 2, there are many different approaches to XML document clustering. However, most of the mentioned algorithms share a common high-level architecture, which consists of three phases [2]:

1. Data representation
2. Similarity computation
3. Document clustering

In the first phase, documents are transformed into a chosen representation. Next, all of the documents are compared according to a chosen similarity measure. Finally, the documents are grouped into clusters with respect to the computed similarity between them.

The above-mentioned clustering methodology is followed by countless algorithms. However, in the context of pattern-based clustering, this methodology is inappropriate, since in pattern-based approaches, documents are assigned to clusters based on global information encapsulated in patterns rather than local information expressed by direct document similarity. Therefore, for the purpose of pattern-based clustering, we propose a new XML document clustering framework, called *XPattern*. In Sect. 3.1, we will present a conceptual description of the framework, and in Sect. 3.2, we will formally define its elements.

3.1 Conceptual description

The XPattern framework consists of four steps:

1. Data transformation
2. Pattern mining
3. Pattern clustering
4. Document assignment

This approach is based on a real-life observation that objects—for instance people—differ from each other in many details. Thus, we say that each person is unique. But if we omit characteristic features, we can see that people manifest similar patterns of behavior that allow us to classify them into a limited amount of profiles (for example, extroverts and introverts).

Following this analogy, we have to define what kind of patterns we are looking for, i.e., decide whether we want to group people according to their behavior or maybe their appearance. Next, we have to find all the patterns that appear in the dataset. Since we do not know how many patterns we will find and there can be even more patterns than people, we have to group the patterns into profiles. Once we have formed an optimal number of profiles, we can simply assign each person to the profile she/he fits best. Let us now discuss each step of the XPattern framework in more detail.

3.1.1 Step 1: Data transformation

The purpose of the first step is to transform input data into a representation that allows for efficient pattern mining according to a chosen pattern definition. The framework is independent of any particular representation, and thus, XML documents can be transformed into any of the data structures discussed in Sect. 2, such as bags of tags or trees.

3.1.2 Step 2: Pattern mining

In the pattern mining step, the transformed input dataset is mined for patterns. A pattern can be defined as any piece of information obtainable from a single document that has a user-specified property, e.g., is unique or appears frequently across the dataset. Typical structural patterns include frequent subtrees and tags, but also features such as tag counts, paths, or statistical measures can be used. As we are describing an abstract framework, we will not use any concrete pattern definition in the remainder of this section and discuss the consequences of particular definitions in Sect. 5.2.

A pattern definition implies a method, which needs to be employed for pattern mining, e.g., patterns defined as frequent subtrees require an Apriori-like algorithm, while patterns defined as distinct tag counts require a simple dataset scan. Since in our framework patterns serve as cluster representatives, the mining algorithm has to produce at least as many patterns as there are expected clusters. That is why, if there are less patterns than expected clusters, we have to restart the pattern mining step and search for new patterns.

3.1.3 Step 3: Pattern clustering

In the third step of XPattern, patterns are clustered into groups, called profiles. The framework does not define any pattern similarity measure necessary for clustering. Two basic approaches arise—similarity measures that compare pattern structures and similarity measures that compare patterns by the number of the documents they co-occur in. The first approach promotes structurally cohesive profiles and takes into account relations between patterns rather than between documents. The second approach promotes profiles with commonly co-occurring patterns and, thus, indirectly uses inter-document relationships. Additionally, the second approach requires less processing. Comparing pattern structures would require calculating similarity between each pair of patterns, while all information needed for clustering using document co-occurrences can be gathered during the pattern mining step. That is why, we propose to use the second approach.

3.1.4 Step 4: Document assignment

After patterns are grouped into profiles, in the final step of the framework, we have to assign all documents to the clusters represented by these profiles. It is performed by testing each document against each profile to check how well the patterns in that profile describe that particular document. A document is assigned to the cluster for which this test produces the best result. In this step, similarly as in the third step, we can use the earlier acquired information about occurrences of patterns in documents. Therefore, we do not need to check whether a document contains a pattern. It is also worth noticing that some documents may not contain any pattern and, thus, may be left unassigned. This feature relates to the problem of outlier detection, as documents without corresponding patterns naturally form a set of outliers.

Let us now focus on the main differences between the steps of XPattern and the earlier mentioned typical clustering methodology [2]. Although the first step of both methodologies transforms objects into a chosen representation, in the traditional approach this is done to compare documents with each other, while in our approach it facilitates the process of pattern mining. Pattern mining and pattern clustering are steps distinctive for our framework. In these steps, we aim at tackling the challenge of creating easily interpretable cluster representatives in the form of profiles. The final step of our framework assigns each document to a profile and, thus, incrementally creates clusters. This differs from the traditional approach where the last step consists of clustering by direct document similarity, which uses only local information and usually cannot be performed incrementally. Finally, it is worth noticing that the XPattern framework facilitates outlier treatment, as it naturally captures documents, which do not fit to any profile.

3.2 Formal definition

The first step of the framework aims at transforming all documents from a dataset $\mathcal{D}$ into a representation that permits the chosen type of feature extraction. We will denote the transformed dataset as $\mathcal{D}^T$.

Definition 1 A *feature* f is any piece of information that can be extracted or calculated from an XML document, e.g., a subtree, element label, or total number of elements. We denote the set of all features by $\mathcal{F}$.

Definition 2 A document $d \in \mathcal{D}$ *contains* a feature f if f can be obtained from its transformed representation $d^T \in \mathcal{D}^T$. We denote the containment relation by $f \in d$ and the number of occurrences of f in d by $m(f, d)$.

Definition 3 A feature f is called a *pattern* if it fulfills a user-specified predicate $pattern(f)$. We denote a single pattern by p and a set of all patterns by $\mathcal{P}$:

$$\mathcal{P} = \{f \in \mathcal{F} : pattern(f)\}$$

The second step of the framework mines all available patterns $\mathcal{P}$ from the dataset, preserving the information about all document-pattern containment relations.

Definition 4 A *set of profiles* Π^k is a set of k sets of patterns defined as follows:

$$\Pi^k = \left\{\pi_1, \pi_2, \ldots, \pi_k : \forall_{i=1\ldots k}\pi_i \neq \emptyset \wedge \pi_i \subseteq \mathcal{P} \wedge \forall_{j=1\ldots k; j\neq i}\pi_i \cap \pi_j = \emptyset\right\}$$

Definition 5 A *profile* π_i is an element of a set of profiles Π^k that represents a cluster c_i.

In the third step of the XPattern framework, the patterns $\mathcal{P}$ are grouped into a set of k profiles Π^k. One can use any algorithm or similarity measure to create the profiles—they are only restricted by the definition of a profile and a profile set (Definitions 4 and 5).

Definition 6 A document d is *connected* to a profile π_i if it contains at least one pattern from that profile:

$$d \sim \pi_i \iff \exists_{p\in\pi_i} p \in d$$

Definition 7 The degree in which a document and a profile are connected (document-profile similarity) is measured by the *connection strength* function *str*, defined as follows:

$$str : \mathcal{D} \times \Pi^k \rightarrow \mathbb{R}_0^+$$

Definition 8 A document d is assigned to a cluster c_i if it has the highest connection strength with profile π_i:

$$c_i = \{d \in \mathcal{D} : \underset{\pi\in\Pi^k}{\arg\max}\; str(d, \pi) = \pi_i\}$$

In the last step of the framework, each document is assigned to a cluster according to Definition 8.

In this section, we have defined a general framework for clustering XML documents by patterns. One can use any document representation, pattern definition, pattern similarity measure, and pattern clustering method to create an algorithm tailored to a specific problem. In the next section, we will present one of the possible instances of the XPattern framework.

4 The PathXP algorithm

In this section, we present a concrete instantiation of the XPattern framework—the PathXP algorithm. Section 4.1 describes this algorithm in detail and illustrates it with an example. Section 4.2 presents an attempt to create a parameterless version of the PathXP algorithm, which automatically estimates the minimum support value as well as the number of clusters.

4.1 The algorithm

Based on the XPattern framework, we propose an algorithm called PathXP, outlined in Algorithm 1, which uses XML paths as features. In PathXP, a feature f will be called a pattern if it is a *maximal frequent* path, i.e., if it is contained in at least *minsup* percent of documents in $\mathcal{D}$, where $minsup$ is a user-defined minimum support parameter, and f is not a subpath of any other frequent path:

$$frequent(f) \iff \exists_{\mathcal{D}' \subseteq \mathcal{D}} \forall_{d' \in \mathcal{D}'} f \in d' \wedge \frac{|\mathcal{D}'|}{|\mathcal{D}|} \geq minsup \tag{1}$$

$$pattern(f) \iff frequent(f) \wedge \neg\exists_{f' \in \mathcal{F}} \left(frequent(f') \wedge f \subset f'\right) \tag{2}$$

The algorithm begins with setting the initial value of $minsup$ at $1/k$ (the value $1/k$ will be discussed in detail in Sect. 4.2), where k is the number of expected clusters. Next, the input dataset $\mathcal{D}$ is mined for maximal frequent paths. Until the number of discovered paths is greater or equal to the number of expected clusters, $minsup$ is divided by 2 and the mining process is restarted. The obtained set of patterns $\mathcal{P}$ is later grouped into k profiles using complete link AHC algorithm with a pattern similarity measure defined as:

$$sim(p_1, p_2) = \frac{\sum_{d \in \mathcal{D}} \min\{m(p_1, d), m(p_2, d)\}}{\sum_{d \in \mathcal{D}} m(p_1, d) + m(p_2, d) - \min\{m(p_1, d), m(p_2, d)\}} \tag{3}$$

Finally, each document is assigned to the profile with which it has the highest connection strength. In PathXP, we define connection strength as the number of patterns contained in a document d which are present in a profile π_i, divided by the size of π_i:

$$str\,(d, \pi_i) = \frac{\sum_{p \in \pi_i} m(p, d)}{|\pi_i|} \tag{4}$$

It is worth noting that dividing by the profile's size is used to eliminate the effect of promoting large profiles.

Algorithm 1 The PathXP clustering algorithm

```
Input: set of XML documents $\mathcal{D}$, number of clusters $k$
Output: set of $k$ clusters $\mathcal{C}$

 1: $minsup \leftarrow \frac{1}{k}$;
 2: $\mathcal{P} \leftarrow AprioriPaths(\mathcal{D}, minsup)$;
 3: while $|\mathcal{P}| < k$ do
 4:   $minsup \leftarrow minsup/2$;
 5:   $\mathcal{P} \leftarrow AprioriPaths(\mathcal{D}, minsup)$;
 6: end while
 7: $\Pi^k \leftarrow AHC(\mathcal{P}, k)$;
 8: $\mathcal{C} \leftarrow k$ empty clusters each defined by one profile $\pi_c \in \Pi^k$;
 9: for all $d \in \mathcal{D}$ do
10:   $bestCluster \leftarrow$ the first profile;
11:   $bestMatchCount \leftarrow 0$;
12:   for all $c \in \mathcal{C}$ do
13:     $matchCount \leftarrow \sum_{p: p \in \pi_c \wedge p \in d} NumOfOccurrences(p, d)$;
14:     if $matchCount > bestMatchCount$ then
15:       $bestMatchCount \leftarrow matchCount$;
16:       $bestCluster \leftarrow c$;
17:     end if
18:   end for
19:   add $d$ to $bestCluster$;
20: end for
```

Table 4 XPattern components defined in PathXP

XPattern	PathXP
Document representation	Set of paths
Pattern definition	Maximal frequent paths
Pattern clustering algorithm	Complete link AHC
Pattern similarity	$sim(p_1, p_2) = \frac{\sum_{d \in \mathcal{D}} \min\{m(p_1,d), m(p_2,d)\}}{\sum_{d \in \mathcal{D}} m(p_1,d) + m(p_2,d) - \min\{m(p_1,d), m(p_2,d)\}}$
Connection strength (document-profile similarity)	$str\,(d, \pi_i) = \frac{\sum_{p \in \pi_i} m(p,d)}{\lvert\pi_i\rvert}$

As an instance of XPattern, PathXP defines all the required components of the framework. Table 4 summarizes all the defined components.

Let us analyze the worst-case complexity of the PathXP algorithm. The problem of mining frequent itemsets is known to be NP-Hard [43]. The theoretical cost of mining frequent paths from a dataset of n documents with m distinct tags is $O(2^m n)$. The pattern clustering step uses *AHC*, hence, for p patterns and k expected clusters, the complexity of this step is $O(k \cdot p^2)$. Finally, the document assignment step searches for all the occurrences of each pattern in each document. A single search for pattern occurrences in a document requires $O(v)$ operations, where v is the number of vertices in the document. Thus, the pessimistic complexity of the document assignment step is $O(p \cdot n \cdot max(v))$, where $max(v)$ is the number of vertices in the longest document in the dataset. Given the above, the overall worst-case complexity of PathXP equals $O(2^m n)$. Since the number of distinct labels m is usually bounded and n is the number of documents in a dataset, this shows that our algorithm can in practice scale linearly up to very large datasets (as evidenced by the scalability test performed in

Sect. 5). Furthermore, since the most costly operation takes place in the pattern mining step, our solution will prove particularly effective in incremental clustering scenarios, where new documents are added to existing clusters.

Example 3 Let us now analyze the operation of PathXP on the example set of documents from Table 1. First, the initial value of $minsup$ is calculated. For the expected number of clusters $k = 2$, minimum support is set to $\frac{1}{2}$. This means that for a dataset of eight documents, a feature must occur in at least four documents to be considered as frequent. Table 5 presents maximal frequent paths (patterns) found in the example dataset.

Since the number of obtained patterns is greater than the number of expected clusters, it is not necessary to decrease $minsup$ and restart frequent path mining. Therefore, the algorithm proceeds with grouping patterns into profiles according to Eq. 3. The result of this operation is presented in Fig. 3.

Table 6 presents the distance matrix created based on the pattern-document co-occurrences. According to this matrix, the complete link AHC algorithm will create two profiles: the first one (π_1) will contain patterns p_1 and p_2, while the second one (π_2) patterns p_3, p_4, p_5 and p_6. It is worth noting that by looking at paths in profiles π_1 and π_2 one can easily interpret the characteristic features of each cluster.

Finally, documents are assigned to profiles. Each document is assigned to the profile with which it has the highest connection strength (see Eq. 4). As Table 7 shows, according to the number of pattern-document co-occurrences, documents d_1, d_2, d_3 and d_4 are assigned to the cluster represented by profile π_1, while documents d_5, d_6, d_7 and d_8 are assigned to the cluster represented by profile π_2. This concludes the algorithm.

As described in Sect. 1.1, choosing a proper pattern definition is an important task. The use of frequent paths as patterns in the PathXP algorithm was dictated by a series of tests (described in Sect. 5) conducted on several different definitions. While subtrees preserve more structural information than paths, the latter are easier to obtain, and, thus, the whole clustering process is more efficient.

Table 5 Patterns found in the documents from Table 1

Id	Pattern
p_1	paper/booktitle/title
p_2	paper/booktitle/year
p_3	volume
p_4	paper/authors/author
p_5	paper/journal/title
p_6	paper/journal/year

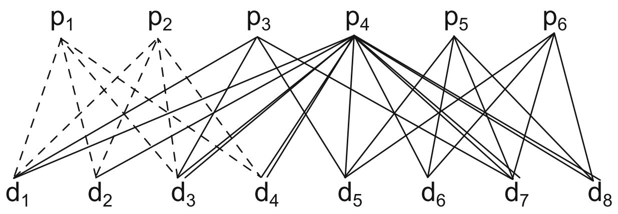


Fig. 3 Occurrences of patterns in the example documents

Table 6 Patterns' distance matrix

	p_1	p_2	p_3	p_4	p_5	p_6
p_1	0.0					
p_2	0.7	0.0				
p_3	0.7	1.0	0.0			
p_4	0.7	0.0	1.0	0.0		
p_5	0.7	1.0	0.0	1.0	0.0	
p_6	0.5	0.5	0.5	0.5	0.5	0.0

Table 7 Connection strengths between documents and profiles

	π_1^2	π_2^2
d_1	**1.0**	0.5
d_2	**1.0**	0.3
d_3	**1.0**	0.8
d_4	**1.0**	0.5
d_5	0.0	**1.0**
d_6	0.0	**0.8**
d_7	0.0	**1.3**
d_8	0.0	**1.0**

Bold values indicate the profile to which the document is assigned

4.2 Parametrization

As stated in the introduction of this paper, choosing a proper minimum support threshold can be a problematic task. That is why, in PathXP we only require user to provide the number of clusters and we set the minimum support threshold automatically. Given the number of clusters k, we propose to set the minimum support parameter to $1/k$. The assumption behind this approach is that documents in each cluster share common patterns; therefore, assuming a uniform document distribution among clusters, the $1/k$ value should allow the algorithm for discovering these patterns. For highly imbalanced datasets, this approach may not be sufficient; however, this problem is out of the scope of this paper and is planned as a future work.

The requirement of knowing the number of clusters a priori is commonly assumed in most XML clustering algorithms. Recent XML clustering surveys [2,32] reveal that nearly all of the approaches proposed so far rely on this assumption. However, such a requirement may discredit the algorithm in many real-world applications. To address this problem, we propose a heuristic approach which automatically detects the number of clusters. The pseudocode of this heuristic is illustrated in Algorithm 2. This approach consists of two phases. In the first phase (lines 1–9), we determine the number of clusters by incrementally iterating through consecutive values of k and triggering our basic PathXP algorithm for each of these values. Once PathXP returns a result with at least one of the clusters empty, the iteration stops and we assume that the previous value of k was the correct number of clusters. After obtaining k, in the second phase (line 10), we simply use this information to cluster the dataset with PathXP.

This approach is based on an intuition that after reaching a certain number of clusters, profiles start to disintegrate and the connection strength between documents and these profiles weakens. This disintegration means that the most unsuited patterns of the weakest profile form a separate profile. This profile should, eventually, get overwhelmed by other, more cohesive profiles, and be left with no documents assigned. This means that this profile is too

Algorithm 2 Parameterless version of PathXP

Input: set of XML documents $\mathcal{D}$
Output: set of clusters $\mathcal{C}$

```
1: $k \leftarrow 1$;
2: while $k < |\mathcal{D}|$ do
3:   $\mathcal{C} \leftarrow PathXP(\mathcal{D}, k)$;
4:   if $\mathcal{C}$ contains an empty cluster then
5:     $k \leftarrow k - 1$;
6:     break;
7:   end if
8:   $k \leftarrow k + 1$;
9: end while
10: $\mathcal{C} \leftarrow PathXP(\mathcal{D}, k)$;
```

weak to stand on its own and should be a part of another profile; thus, the number of clusters should be decreased.

Experiments show that the unparameterized version of our algorithm can produce results of similar quality as PathXP, however, does not always accurately predict k with more difficult datasets. Details will be discussed in Sect. 5.

5 Experimental evaluation

The proposed algorithm was evaluated in several experiments to inspect its properties and compare it with competitive approaches. In the following subsections, we describe all of the used datasets, discuss experimental setup, and analyze experiment results.

5.1 Datasets and experimental setup

The proposed algorithm was tested against 5 real and 2 synthetic datasets, summarized in Table 8. For real data, we chose one heterogeneous dataset from the *SIGMOD Record* [36] (sig) and four homogeneous datasets from the 2005/2006 INEX competition (db$N$, $N = 0 \ldots 3$), which are drawn from the IMDB movie database and ranked according to their difficulty—the higher the N, the more overlap there is between the classes. To generate synthetic datasets, we used the *ToXgene* framework [3] with two sets of schemas: one containing 10 different DTDs for generating a heterogeneous dataset (het), second containing three similar DTDs for generating a homogeneous dataset (hom). The parameter indicating the maximal number of times that a node can appear as a child of its parent node was set to 4 for all of the synthetic datasets.

Table 8 Characteristic of datasets

Dataset	Classes	Number of documents	Avg. size	Avg. width	Avg. height	Distinct labels
SIGMOD (sig)	2	140	82.66	32.16	5.46	39
INEX (db0-3)	11	4,825	220.03	112.24	5.32	195
Heterogeneous (het)	10	1,000	40.11	21.82	3.98	73
Homogeneous (hom)	3	300	36.37	18.07	4.00	12

All of the compared algorithms were implemented in the C# programming language. We have also used a C++ implementation of the CMTreeMiner [9] algorithm for mining maximal frequent subtrees. The experiments took place on a computer with a 2,80 GHz Inter Core i7 processor and 16 GB of RAM.

As in [1], to evaluate clustering quality, we used precision and recall:

$$precision = \frac{\sum_i s_i}{\sum_i s_i + \sum_i v_i}, \quad recall = \frac{\sum_i s_i + \sum_i v_i}{|\mathcal{D}|}, \quad i = 1 \ldots k$$

where k is the number of clusters, $|\mathcal{D}|$ is the number of documents in the dataset, s_i is the number of documents correctly assigned to the i-th cluster and v_i is the number of documents incorrectly assigned to the i-th cluster. Although these evaluation measures originate from supervised learning, they can be used in our setting as all of the employed datasets contain labeled documents.

5.2 Alternative pattern definitions

Before discussing the properties of PathXP, let us analyze the impact of using various pattern definitions with the XPattern framework. As mentioned earlier, we propose to use XML paths, as they offer a compromise between full structural information and lightweight processing. This decision is supported by experimental results given in Table 9, which presents precision, recall, and clustering time of PathXP using subtrees, paths, tags, and metadata as patterns.

Subtrees are often presented as objects that best summarize structural information of an XML document. They include all the information contained in tags or paths and provide additional sibling information. As experimental results show, subtrees are the most expensive pattern definition from the compared set. Frequent subtrees are usually larger than tag or path patterns and, thus, more resource consuming. Furthermore, as stated by Chi et al. [9], the cost of mining maximal frequent subtrees is linearly proportional to the depth h of a document

Table 9 Precision, recall, and clustering time for PathXP with different pattern definitions

Pattern	sig	het	hom	db0	db1	db2	db3
	Precision						
Subtrees	1.00	1.00	0.90	–	–	–	–
Paths	1.00	1.00	0.92	0.66	0.71	0.66	0.64
Tags	0.51	1.00	0.35	0.73	0.69	0.45	0.44
Metadata	0.98	0.45	0.36	0.22	0.18	0.15	0.17
	Recall						
Subtrees	1.00	1.00	1.00	–	–	–	–
Paths	1.00	1.00	1.00	1.00	0.99	0.99	1.00
Tags	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Metadata	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Clustering time [s]						
Subtrees	0.07	1.40	3.24	–	–	–	–
Paths	1.57	0.09	0.17	65.25	137.76	311.64	403.38
Tags	0.08	1.06	0.05	21.44	25.62	42.66	44.07
Metadata	0.02	0.03	0.01	0.18	0.24	0.40	0.38

and exponentially proportional to its width w. This gives a worst-case complexity of $O(2^w)$ just for the pattern mining step. For this reason, although accurate for smaller datasets, this pattern definition cannot be used to cluster large, real-world datasets such as the INEX movie database. In our study, experiments involving subtrees as patterns on datasets db0-3 consumed all available memory before providing results, which is illustrated by blank values in Table 9.

Simple sequences of tags are much easier to process than document trees. As time results presented in Table 9 show, tag patterns are among the easiest to acquire and process. This result finds its confirmation in the complexity analysis of tag mining, which in the worst-case scenario requires $O(M \cdot n)$ operations, where n is the number of documents in a dataset and M is the number of tags in the largest document in the dataset. However, precision acquired for the tag-based approach shows that tag patterns performed well only for the synthetically generated heterogeneous dataset (het) and for the noiseless real dataset (db0). This is understandable as single tags can characterize very distinct object groups. For more difficult homogeneous datasets (db1–3, hom), where most objects are very similar to each other, as well as for real heterogeneous dataset (sig), tags do not convey enough structural information to perform proper clustering. Additional data are needed for these datasets, such as additional structural information or metadata.

Alternatively, pattern definitions constructed from document structures can be replaced by data describing these structures, i.e., by structural metadata. In order to determine how well metadata can capture document characteristics, first, we identified 10 different structural summaries: number of elements, number of distinct elements, number of levels, average number of elements at all levels, standard deviation of number of elements at all levels, number of leafs, average path length, standard deviation of path length, average number of children for all nodes, and standard deviation of number of children for all nodes. Next, for each dataset, we performed tests of all 1,023 possible combinations of these parameters. The combination that performed best was based on two very basic structural summaries: the number of distinct elements and the number of levels in a tree. The quality and time evaluation of this combination are presented in Table 9 (Metadata). As results indicate, using metadata can produce results of a competing quality compared to the tag-only approach for heterogeneous (sig, het) and synthetic homogeneous (hom) datasets, while requiring much less time. However, real homogeneous datasets (db0-3) reveal that this approach can only be used for simple problems or as additional information for other clustering methods.

Looking at the results a question arises: to what extent structural metadata can describe documents? For a closed set of XML data sources, documents can be clustered using solely structural metadata, but for very large datasets, it seems more reasonable to use metadata only as additional clustering information. As denoted by Halevy et al. [20], recent research shows that document processing requires the use of all available data and for this reason, the use of metadata such as document statistics can possibly become an important part of a real-world pattern definition, but as the presented results show, it is not sufficient to use them alone.

This paper focuses mainly on clustering XML documents by structure, but it is worth noting that the presented approach can work equally well with textual data as patterns. One can use n-grams, synsets, or other word relations as cluster representatives. Since using patterns for clustering is a general idea, which does not imply their specific definition, it can be used with content as well as structure-based document representations. However, since XML clustering has many domains of application, we think that algorithms for this problem should give the user a choice on which information should be taken into account. For example, in domains such as chemistry, compounds can be encoded only by the structure of an XML format, thus eliminating the necessity for textual analysis. On the other hand, clustering records of a consistent, XML-based employee database can be performed without structural information. In other cases, both structure and content should be taken into account.

5.3 Analysis of the components of the proposed algorithm

While constructing the PathXP algorithm, we analyzed several possibilities concerning algorithm functioning and parametrization. In this section, we present results of this analysis. First, we investigate how restricting frequent patterns to maximal frequent patterns, counting multiple pattern occurrences, and considering pattern uniqueness influence clustering quality. Secondly, we analyze the impact of limiting the length of path patterns. Finally, we perform a parameter sensitivity test and a scalability test of the proposed algorithm.

Let us start by analyzing three binary algorithm settings:

- **C**: Counting multiple occurrences of patterns in documents.
  This information is used by function $m(f, d)$ from Definition 2 to calculate pattern similarity and connection strength, as presented in Eqs. 3 and 4. In order to discard this information, we alter the definition of $m(f, d)$, so that it produces a binary result: 1 when $f \in d$, 0 otherwise.
- **W**: Weighting patterns according to their uniqueness.
  This information is expressed by the following formula:

$$Weighted\ Support(p) = \sum_{d:p \in d} \frac{1}{|\{p_i : p_i \in d\}|},$$

  When used, it is embedded into the connection strength formula (Eq. 4) as follows:

$$str\ (d, \pi_i) = \frac{\sum_{p \in \pi_i} m(p, d) \cdot Weighted\ Support(p)}{|\pi_i|}$$

- **M**: Using only maximal frequent paths as patterns.
  This information is used in the $pattern(f)$ predicate in Eq. 2. If we want to use all frequent paths, the $pattern(f)$ predicate simply changes to:

$$pattern(f) \iff frequent(f)$$

Table 10 presents the precision obtained by the algorithm for different combinations of settings {C, W, M}. Recall was omitted, as it remained unchanged for each dataset.

In order to determine whether the settings significantly influence the algorithms quality, for every dataset we ranked each algorithm's performance from 1 to 8, where 1 is the highest and 8 is the lowest score. In case when one or more algorithms were tied, average ranks were assigned (e.g., if two algorithms were tied at the third place, each was granted a rank of 3.5). Once created, the ranking was used to perform a Friedman test [11]. The null hypothesis for this test is that there is no difference between the performance of all the tested algorithm settings. Moreover, in case of rejecting this null hypothesis, we use the Bonferroni–Dunn post hoc test [11] to verify whether the performance of the best setting is statistically different from the remaining approaches. The result of the test is visualized in Fig. 4. Assuming a Chi-square distribution with 7 degrees of freedom, the value of Chi-square for the Friedman test equals 15.30. With the significance level of $\alpha = 0.05$, we can reject the null hypothesis, which indicates that the algorithm settings are not identical. Furthermore, the Critical Difference (CD) chosen by the Bonferroni–Dunn test $CD = 3.5$ indicates that settings 101 and 100 perform significantly better than setting 010. This means that combining information about

Table 10 Precision for varying algorithm settings

Settings			Precision							Avg. rank
C	W	M	sig	het	hom	db0	db1	db2	db3	
0	0	0	1.00	1.00	1.00	0.62	0.54	0.51	0.47	4.9
0	0	1	1.00	1.00	1.00	0.54	0.51	0.54	0.49	5.0
0	1	0	1.00	1.00	0.76	0.49	0.28	0.17	0.18	6.7
0	1	1	1.00	1.00	0.76	0.61	0.54	0.53	0.50	5.4
1	0	0	1.00	1.00	0.92	0.73	0.68	0.58	0.55	3.1
1	0	1	1.00	1.00	0.92	0.66	0.71	0.66	0.64	2.6
1	1	0	0.50	1.00	0.51	0.72	0.63	0.55	0.52	4.9
1	1	1	1.00	1.00	0.79	0.65	0.70	0.65	0.62	3.4

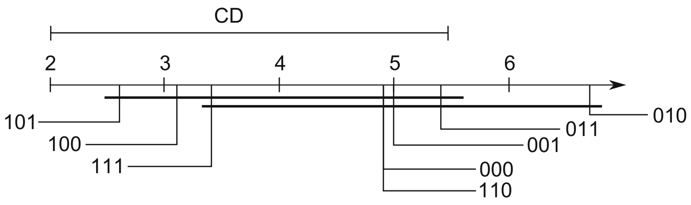


Fig. 4 Friedmann test performed on the results from Table 10

Table 11 Precision for varying maximal path length

Maximal path length	Precision							Avg. rank
	sig	het	hom	db0	db1	db2	db3	
1 (tags)	0.51	1.00	0.35	0.73	0.69	0.45	0.44	3.6
2 (edges)	0.79	1.00	0.35	0.66	0.66	0.49	0.44	4.0
3	1.00	1.00	0.96	0.67	0.68	0.58	0.55	2.6
4	1.00	1.00	0.92	0.66	0.69	0.61	0.59	2.6
≥ 5	1.00	1.00	0.92	0.66	0.71	0.66	0.64	2.1

multiple pattern occurrences with maximal patterns is better than weighting patterns by their uniqueness combined with using all patterns instead of maximal. Additionally, analyzing the results of the two best settings (101 and 100), we observe that setting 101 performs equally good or better on all datasets, except for db0. That is why, in PathXP we use the combination of counting multiple pattern occurrences with maximal patterns.

Apart from analyzing different pattern counting schemes, we analyzed the possibility of limiting the maximal length of paths used as patterns. The results of this study are illustrated in Table 11.

From the computational point of view, the most desired path length is 1 (tags), as longer paths yield exponential worst-case complexity in the pattern mining process. In order to determine whether the path length changes algorithm quality, we performed another Friedman test illustrated in Fig. 5. This time we have five algorithms with path lengths ranging from 1 to 5 (the paths of length 5 and above produced the same results). For this setting, the Friedman

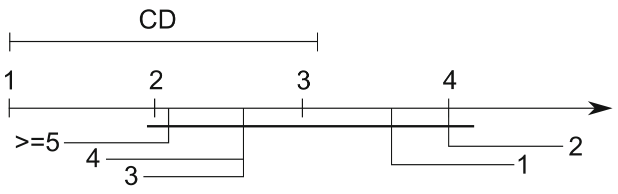


Fig. 5 Friedmann test performed on the results from Table 11

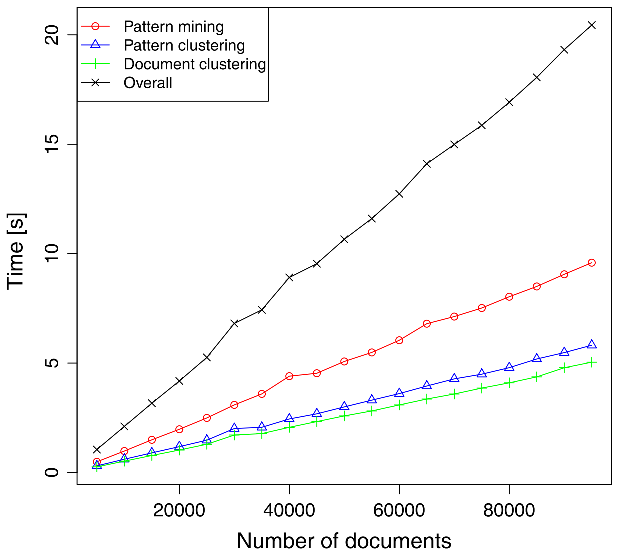


Fig. 6 Scalability test results for PathXP

test does not reveal a significant difference between the path lengths ($\alpha = 0.05$). However, because paths of unlimited length provided the best result for most of the datasets, in PathXP we are using frequent paths of unlimited length as patterns.

Finally, we performed a scalability and sensitivity test to verify how PathXP works for different dataset sizes and parameter values. For scalability test, we generated 20 heterogeneous datasets containing from 5,000 to 100,000 documents. As results presented in Fig. 6 show, the most time-consuming stage of our algorithm is pattern mining, which is expected, as the worst-case complexity of this step is $O(2^m n)$, where m is the number of distinct tags in the dataset of n documents. Additionally, the plot confirms that with the increasing number of documents, the execution time of our algorithm increases linearly.

To perform the parameter sensitivity test, we used the first dataset from the previously mentioned INEX competition (db0). This test was performed for $minsup$ varying between 0.01 and 0.3, as higher values yield too few patterns to form a required number of clusters. Figure 7 contains plots showing how minimum support changes the execution time of all steps in our algorithm. Similarly to the scalability test, the execution time depends mainly on pattern mining. The plot shows that with decreasing values of the minimum support parameter, execution time increases exponentially. This exponential growth concerns also the pattern clustering and document assignment. However, these steps of our algorithm do

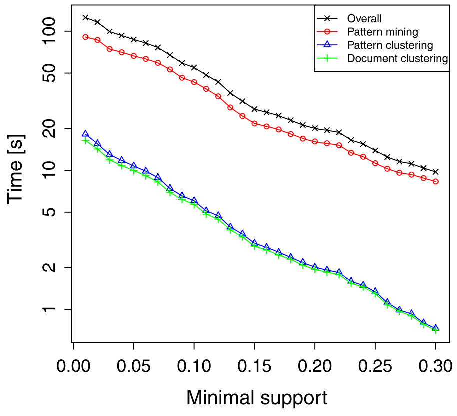


Fig. 7 Sensitivity test results for PathXP with respect to minimal support: time

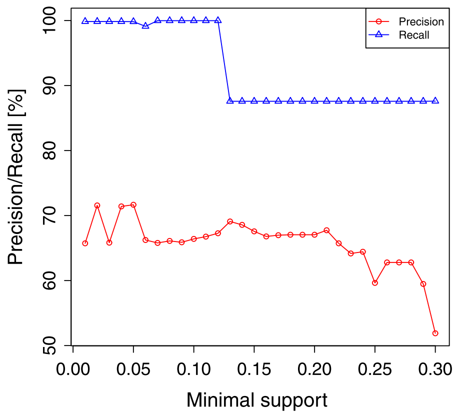


Fig. 8 Sensitivity test results for the PathXP algorithm with respect to minimal support: precision and recall

not use the minimum support parameter; therefore, the shapes of the plots in fact illustrate the growth in number of patterns obtained during pattern mining.

Apart from time sensitivity, we have also analyzed how the minimum support parameter influences the quality of clustering. The outcome of this analysis is presented in Fig. 8. As expected, the overall clustering quality decreases with the increase in the minimum support value. This is due to the fact that higher $minsup$ values produce more general patters, which are not discriminative enough to distinguish between different clusters. However, it is worth noting that for a wide range of $minsup$ values (up until 0.21) precision holds a steadily high level (66–73 %). Only after the 0.21 mark, the quality drops notably. Similarly, recall remains steady at approximately 100 % for $minsup$ between 0 and 0.13 and drops to 88 % above the 0.13 threshold. This means that for $minsup > 0.13$, approximately 12 % of the documents have no corresponding patterns in any profile. In other words, after $minsup$ reaches a certain threshold, the patterns generated in the mining process begin to reflect only the most common information in the dataset. As a result, in the document assignment step, the documents with less common characteristics have no matching patterns and, therefore, are not assigned to any cluster. This, in turn, results in the lower recall value.

The parameter sensitivity test (Figs. 7 and 8) showcases that our solution is predictably sensitive to the value of minimum support in terms of both execution time and clustering quality. The lower the parameter's value, the longer the execution time, but also higher clustering quality. It is worth noticing that our suggestion of setting the value of this parameter to $1/k$ ($k_{db0} = 11$) produces a compromise with satisfying quality and acceptable execution time.

5.4 Comparative study of clustering algorithms

After establishing the properties of PathXP, a set of experiments was conducted to compare the newly proposed algorithm against four competitive structure-based algorithms. The results of this comparison are presented in Table 12 (precision for algorithms other than PathXP was taken from the Report on the XML Mining Track at INEX 2005 and INEX 2006 [12]). The PathXP settings used for the comparison were default: counting multiple pattern occurrences with maximal, unlimited frequent paths as patterns.

The comparison results are inconclusive, as they do not clearly indicate which approach is best. PathXP produced equally good or better results across all datasets compared to Vercoustre's et al., and Nayak's and Xu's approaches. On the other hand, Candillier's and Hagenbuchner's approaches outperformed PathXP on the first dataset (db0). Unfortunately, results for the remainig datasets (db1–3) were unavailable due to reasons not clearly stated by authors [8,18]. Given the above, we cannot significantly state that our algorithm performs better than the others. However, we presented a competitive solution, which additionally addresses the challenges stated in the introduction of this paper.

Apart from comparing PathXP with other approaches, we also evaluated its performance against our parameterless algorithm proposed in Sect. 4.2. Table 13 presents the number of clusters, precision, recall, and clustering time of PathXP and parameterless PathXP.

Table 12 Comparison of PathXP with other structure-based algorithms

Algorithm	Precision			
	db0	db1	db2	db3
Vercoustre et al. [39]	0.45	0.71	0.66	0.53
Candillier et al. [8]	0.78	–	–	–
Hagenbuchner et al. [18]	0.97	–	–	–
Nayak and Xu [29]	0.60	0.60	0.59	0.59
PathXP	0.66	0.71	0.66	0.64

Table 13 Comparison of PathXP and parameterless PathXP

Algorithm	sig	het	hom	db0	db1	db2	db3
	Number of clusters						
PathXP	2	10	3	11	11	11	11
Parameterless	2	11	9	22	23	14	18
	Precision						
PathXP	1.00	1.00	0.92	0.66	0.71	0.66	0.64
Parameterless	1.00	1.00	0.88	0.72	0.74	0.69	0.65
	Recall						
PathXP	1.00	1.00	1.00	1.00	1.00	0.99	1.00
Parameterless	1.00	1.00	1.00	1.00	0.99	0.99	0.99
	Clustering time (s)						
PathXP	2	<1	<1	65	138	312	403
Parameterless	1	19	1	1,538	3,176	3,647	5,999

As the results in Table 13 show, the parameterless version of our algorithm produced clusters of quality comparable to when the number of clusters was provided. For heterogeneous datasets (sig, het), the results are nearly identical—parameterless algorithm found only one more cluster for het dataset than expected. For homogeneous datasets (hom, db0–db3), the number of automatically detected clusters is higher than the expected value. The reason for this overestimation lies in a fact that with homogeneous datasets there is naturally more overlap between the clusters than with heterogeneous datasets. This results in less cohesive profiles and leads to a more uniform distribution of connection strength, even for higher values of k. This, in turn, causes our parameterless algorithm to find empty clusters only after exceeding the actual number of clusters. The additional cost of automatic cluster number detection is apparent in the processing time. For the analyzed datasets, the processing time of the parameterless PathXP is over an order of magnitude higher than the parametrized version. Such an overhead is a direct consequence of repetitive triggering of the PathXP algorithm for each consecutive value of k, until finding an empty cluster.

6 Conclusions and future work

In this paper, we have stated the main challenges concerning XML document clustering by structure and proposed a pattern-based framework called XPattern along with an algorithm called PathXP, which successfully address these challenges. Thanks to the idea of representing each cluster by a set of patterns, the results are clearly identified and easy to interpret (Challenge C1). Additionally, the pattern-based algorithm detects and excludes outliers from the analysis (Challenge C4). We have also proposed a parameterless version of our algorithm, which automatically determines both the number of clusters and the minimum support value (Challenge C3).

By exploring different types of pattern definitions, from simple metadata to complex subtrees, we have discovered that frequent paths provide very good clustering quality while maintaining reasonable efficiency (Challenge C2). We have also shown that the often omitted information about the number of occurrences of a pattern in a single document can significantly improve clustering quality (Challenge C5).

In the future, we plan to continue our research to address the problem of clustering of imbalanced datasets. We would also like to explore possible consequences of using sets of frequent paths other than maximal, by selecting additional, characteristic, non-maximal patterns. Other lines of further research include adding content to the analysis and extending the proposed approach to XML classification.

Acknowledgments This work was partly supported by the Polish National Science Center under Grant No. DEC-2011/01/B/ST6/05169.

References

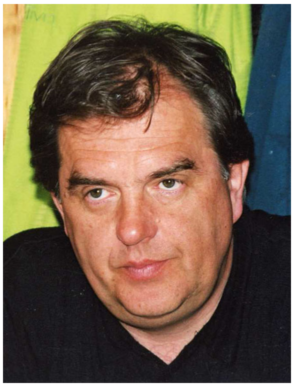
1. Aggarwal CC, Ta N, Wang J, Feng J, Zaki MJ (2007) XProj: a framework for projected structural clustering of xml documents. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, pp 46–55

2. Algergawy A, Mesiti M, Nayak R, Saake G (2011) Xml data clustering: an overview. ACM Comput Surv 43(4):25–41
3. Barbosa D, Mendelzon AO, Keenleyside J, Lyons KA (2002) Toxgene: a template-based data generator for XML. In: Proceedings of the 2002 ACM SIGMOD international conference on management of data, p 616
4. Bennett CH, Gács P, Li M, Vitányi PMB, Zurek WH (1998) Information distance. IEEE Trans Inf Theory 44(4):1407–1423
5. Bray T, Paoli J, Sperberg-McQueen C (1998) Extensible markup language (XML) 1.0. Recommendation, World Wide Web Consortium (W3C). http://www.w3.org/TR/1998/REC-xml-19980210
6. Brzezinski D, Lesniewska A, Morzy T, Piernik M (2011) XCleaner: a new method for clustering xml documents by structure. Control Cybern 40(3):877–891
7. Buttler D (2004) A short survey of document structure similarity algorithms. In: Proceedings of the international conference on internet computing, IC '04, pp 3–9
8. Candillier L, Tellier I, Torre F (2005) Transforming XML trees for efficient classification and clustering. In: Proceeding of the 4th international workshop of the initiative for the evaluation of XML retrieval, INEX 2005, pp 469–480
9. Chi Y, Xia Y, Yang Y, Muntz RR (2005) Mining closed and maximal frequent subtrees from databases of labeled rooted trees. IEEE Trans Knowl Data Eng 17(2):190–202
10. Dalamagas T, Cheng T, Winkel KJ, Sellis TK (2004) Clustering XML documents by structure. In: 3rd Helenic conference on methods and applications of artificial intelligence AI, SETN, LNCS, vol 3025, pp 112–121
11. Demsar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
12. Denoyer L, Gallinari P (2007) Report on the XML mining track at INEX 2005 and INEX 2006: categorization and clustering of XML documents. SIGIR Forum 41:79–90
13. Dolin RH, Alschuler L, Beebe C, Biron PV, Boyer SL, Essin D, Kimber E, Lincoln T, Mattison JE (2001) The HL7 clinical document architecture. J Am Med Inform Assoc 8(6):552–569
14. Doucet A, Ahonen-Myka H (2002) Naïve clustering of a large XML document collection. In: Proceedings of the 1st workshop of the initiative for the evaluation of XML retrieval (INEX), pp 81–87
15. Fayyad UM, Piatetsky-Shapiro G, Smyth P (1996) From data mining to knowledge discovery: an overview. In: Advances in knowledge discovery and data mining, AAAI, pp 1–34
16. Fellbaum C (1998) WordNet: an electronic Lexical database. Bradford Books, Cambridge
17. Flesca S, Manco G, Masciari E, Pontieri L, Pugliese A (2002) Detecting structural similarities between XML documents. In: WebDB, pp 55–60
18. Hagenbuchner M, Sperduti A, Tsoi AC, Trentini F, Scarselli F, Gori M (2005a) Clustering XML documents using self-organizing maps for structures. In: Proceeding of the 4th international workshop of the initiative for the evaluation of XML retrieval, INEX 2005, pp 481–496
19. Hagenbuchner M, Sperduti A, Tsoi AC, Trentini F, Scarselli F, Gori M (2005b) Clustering XML documents using self-organizing maps for structures. In: Workshop of the initiative for the evaluation of XML retrieval
20. Halevy AY, Norvig P, Pereira F (2009) The unreasonable effectiveness of data. IEEE Intell Syst 24(2):8–12
21. Helmer S (2007) Measuring the structural similarity of semistructured documents using entropy. In: Proceedings of the 33rd international conference on very large data bases, pp 1022–1032
22. Kohonen T (2000) Self-organizing maps. Springer series in information science, 3rd edn. Springer, Berlin
23. Kurgan LA, Swiercz W, Cios KJ (2002) Semantic mapping of XML tags using inductive machine learning. In: Proceedings of the 2002 international conference on machine learning and applications, pp 99–109
24. Lee ML, Yang LH, Hsu W, Yang X (2002) XClust: clustering XML schemas for effective integration. In: Proceedings of the 2002 ACM CIKM international conference on information and knowledge management, pp 292–299
25. Lesniewska A (2009) Clustering XML documents by structure. In: Associated workshops and doctoral consortium of the 13th east European conference, pp 238–246
26. Leung HP, Chung KFL, fai Chan SC, Luk RWP (2005) XML document clustering using common XPath. In: 2005 International workshop on challenges in web information retrieval and integration (WIRI 2005), pp 91–96
27. Lian W, Cheung DWL, Mamoulis N, Yiu SM (2004) An efficient and scalable algorithm for clustering XML documents by structure. IEEE Trans Knowl Data Eng 16(1):82–96
28. Nayak R, Iryadi W (2006) XMine: a methodology for mining XML structure. In: Proceedings of the 8th Asia-Pacific web conference, pp 786–792
29. Nayak R, Xu S (2005) XML documents clustering by structures. In: Proceeding of the 4th international workshop of the initiative for the evaluation of XML retrieval, INEX 2005, pp 432–442
30. Nierman A, Jagadish HV (2002) Evaluating structural similarity in XML documents. In: WebDB, pp 61–66

31. Pawlik M, Augsten N (2011) RTED: a robust algorithm for the tree edit distance. PVLDB 5(4):334–345
32. Piernik M, Brzezinski D, Morzy T, Lesniewska A (2014) XML clustering: a review of structural approaches. Knowl Eng Rev. doi:10.1017/S0269888914000216
33. Salton G, McGill M (1984) Introduction to modern information retrieval. McGraw-Hill Book Company, New York
34. Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620
35. Shen Y, Wang B (2003) Clustering schemaless XML documents. In: Proceedings of CoopIS, DOA, and ODBASE 2003, LNCS, vol 2888, pp 767–784
36. SIGMOD (2011) http://www.sigmod.org/publications/sigmod-record/xml-edition
37. Tagarelli A, Greco S (2006) Toward semantic XML clustering. In: Proceedings of the 6th SIAM international conference on data mining, pp 188–199
38. Tekli J, Chbeir R, Yétongnon K (2009) An overview on XML similarity: background, current trends and future directions. Comput Sci Rev 3(3):151–173
39. Vercoustre AM, Fegas M, Gul S, Lechevallier Y (2005) A flexible structured-based representation for XML document mining. In: Proceeding of the 4th international workshop of the initiative for the evaluation of XML retrieval, INEX 2005, pp 443–457
40. W3C (2001) Mathematical markup language (MathML) Version 2.0. W3C Recommendation
41. W3C (2007) SOAP specifications. http://www.w3.org/TR/soap
42. Westbrook J, Ito N, Nakamura H, Henrick K, Berman HM (2005) PDBML: the representation of archival macromolecular structure data in XML. Bioinformatics 21(7):988–992
43. Yang G (2004) The complexity of mining maximal frequent itemsets and maximal frequent patterns. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, pp 344–353
44. Yang J, Cheung WKW, Chen X (2009) Learning element similarity matrix for semi-structured document analysis. Knowl Inf Syst 19(1):53–78
45. Yoon JP, Raghavan V, Chakilam V (2001) BitCube: A three-dimensional bitmap indexing for XML documents. In: Proceedings of the 13th international conference on scientific and statistical database management, pp 158–167

Maciej Piernik received his B.Sc. and M.Sc. degrees in computer science from the Faculty of Computing at Poznan University of Technology in 2009 and 2010, respectively. Currently, he is pursuing the Ph.D. degree at the same university, where he is also holding the position of research assistant. His research interests include data mining and machine learning, with the focus on clustering and classification of semi-structured data and tree similarity.

Dariusz Brzezinski received his B.Sc. and M.Sc. degrees in Computer Science from Poznan University of Technology, Poland, in 2009 and 2010, respectively. He is currently working toward his Ph.D. degree at this university. His research interests include data stream mining, concept drift, online classification algorithms, and XML document clustering.



Tadeusz Morzy is a professor in the Computing Science Department of Poznan University of Technology. He received his M.Sc., Ph.D. and Polish Habilitation from the Technical University of Poznan, Poland. He has authored and coauthored over 100 papers on databases, data mining, and data warehousing. He is co-author of a book on "Concurrency Control in Distributed Database Systems" by North-Holland, editor and coauthor of "Handbook on Data Management" by Springer, and author of "Data Mining: Methods and Algorithms" (in Polish). He served as General Chair of the 2nd and 16th ADBIS Conferences (1998, 2012) and has served/serves on numerous program committees of international conferences and workshops. His research interests include data mining, data warehousing, transaction processing in database and data warehouse systems, access methods and query processing for databases, database optimization and performance evaluation.