UserMap – an Enhancing of User-Driven XML-to-Relational Mapping Strategies
(Technical Report)

Irena Mlynkova

Charles University
Faculty of Mathematics and Physics
Department of Software Engineering
Malostranske nam. 25
118 00 Prague 1, Czech Republic
irena.mlynkova@mff.cuni.cz

Abstract. As XML has undoubtedly become a standard for data representation, it is inevitable to propose and implement techniques for efficient managing of XML data. A natural alternative is to exploit features and functions of (object-)relational database systems, i.e. to rely on their long theoretical and practical history. The main concern of such techniques is the choice of an appropriate XML-to-relational mapping strategy.
In this paper we focus on enhancing of *user-driven* techniques which leave the mapping decisions in hands of users. We propose an algorithm which exploits the user-given annotations more deeply searching the user-specified "hints" in the rest of the schema and applies an adaptive method on the remaining schema fragments. We describe the proposed algorithm, the similarity measure designed for this purpose, sample implementation of key features of the proposal called *UserMap*, and results of experimental testing on real XML data.

1 Introduction

Without any doubt the eXtensible Markup Language (XML) [9] is currently de-facto a standard for data representation and manipulation. Its popularity is given by the fact that the basic W3C recommendations are well-defined, easy-to-learn, and at the same time still enough powerful. The popularity naturally invoked a boom of their efficient implementations based on various storage strategies from the traditional ones such as file systems to brand-new ones proposed particularly for XML structures, so-called *native* approaches or directly *native XML databases.*

Probably the most natural and practically used approach involves techniques which exploit features of (object-)relational database systems, though they are not as efficient as the native ones due to the key problem of structural differences between XML data and relations. The reason for the popularity is that relational databases are still regarded as universal and powerful data processing tools and their long theoretical and practical history can guarantee a reasonable level of reliability and efficiency. Contrary to native methods it is not necessary to start "from scratch" but we can rely on a mature and verified technology, i.e. properties that no native XML database can offer yet. On this account we believe that these methods and their possible improvements should be further enhanced.

Currently there is a plenty of existing works concerning database-based[1] XML data management. Almost all the major database vendors more or less support XML and even the SQL standard has been extended by a new part SQL/XML [14] which introduces new XML data type and operations for XML data manipulation. The main concern of the database-based XML techniques is the choice of an appropriate XML-to-relational mapping strategy, i.e. the way the given XML data are stored into relations. We can distinguish the following three types approaches [3] [19]:

– *fixed* mapping methods based on predefined set of mapping rules and appropriate heuristics (e.g. [12] [25]),
– *adaptive* methods which adapt the target database schema to the intended application (e.g. [16] [8]), and
– methods which leave the mapping decisions in hands of users (e.g. [11] [4]).

The first set of methods can be further divided [3] into *generic* and *schema-driven* ones depending on omitting or exploiting the existence of corresponding XML schema. However, both the types use a straightforward mapping strategy regardless the intended future usage. On the contrary, adaptive methods automatically adapt the database schema of a fixed method to the given additional information. The best known representatives, so-called *cost-driven* methods, usually search a space of possible XML-to-relational mappings and choose the one which conforms to the required application, specified using a sample set of XML data and XML queries, the most, i.e. where the provided queries over the given data can be evaluated most efficiently. Finally, the last mentioned type of methods can be also divided into two groups. We distinguish so-called *user-defined* and *user-driven* methods [19] which differentiate in the amount of necessary user interaction. In the former case a user is expected to define both the target database schema and the required mapping strategy, i.e. to do all the work "manually". In the latter case a default mapping strategy is defined but a user can specify local mapping changes from the predefined set of other allowed mapping strategies, usually using schema annotations. In other words, the user-driven approach solves the main disadvantage of the user-defined one – the requirement of a user skilled in two complex technologies who is, in addition, able to specify an optimal database schema for a particular application. Note that the user-driven techniques can be regarded as a type of adaptive methods too [19], since they also adapt the default target schema to additional information, in particular to user-specified requirements.

[1] In the rest of the paper the term "database" represents an (O)RDBMS.

In this paper we focus on further enhancing of user-driven techniques, particularly on their (in our opinion) two main persisting disadvantages. The first one is the fact that the default mapping strategy is (to our knowledge) always a fixed one. It is quite a surprising finding since we know that the proposed systems are able to store schema fragments in various ways. From this point of view an adaptive enhancing of the fixed method seems to be quite natural and suitable. The second key shortcoming we are dealing with is weak exploitation of the user-given information. We believe that the schema annotations a user provides can not only be directly applied on particular schema fragments, but the information they carry can be further exploited. The main idea is quite simple – we regard the annotations as "hints" how a user wants to store particular XML patterns and we use this information twice again. Firstly, we search for similar patterns in the rest of the schema and store the found fragments in a similar way. And secondly, we exploit the information in the adaptive strategy for not annotated parts of the schema.

To sum up, the main contribution of this paper is a proposal of an algorithm which enhances classical user-driven strategies using the following two approaches:

– a deeper exploitation of the information carried in user-given schema annotations and
– an adaptive mapping strategy for not annotated parts of the schema.

Firstly, we describe the proposed algorithm theoretically. We discuss the key ideas and problems, their possible solutions, and reasons for our particular decisions. Secondly, we describe sample implementation of the proposal called *UserMap* which (among others) involves a similarity measure focussing on structural similarity we have proposed particularly for the purpose of the algorithm. And finally we show and discuss the results of corresponding experimental testing which illustrate the behavior of the proposed algorithm.

The rest of the paper is structured as follows: Section 2 contains a motivation for focusing on user-given information. Section 3 overviews the existing related works in the area of user-driven methods, adaptive methods, and similarity of XML data. In Section 4 we describe and discuss the proposed algorithm in detail. Section 5 describes the sample implementation and its tuning and discusses results of experimental tests on real XML data. Finally, Section 6 provides conclusions and outlines our future work.

2 Motivation

The key concern of our approach is to exploit the user-given information as much as possible. We result from the idea of user-driven enhancing of the user-defined techniques, where a user is expected to help the mapping process, not to perform it. We want to go even farther. But first of all we discuss why user-given information is so important to deal with.

A simple demonstrative example can be a set of XML documents which contain various XHTML [1] fragments. A classical fixed schema-driven mapping strategy (e.g. [25] [18]) would decompose the fragments into a number of relations. Since we know that the standard XHTML DTD allows, e.g., complete subgraphs on up to 10 nodes, the reconstruction of such fragments would be a really expensive operation in terms of the number of join operations. But if we knew that the real complexity of such fragments is much simpler (and the analysis of real XML data shows that it is quite probable [21]), e.g. that each of the fragments can be described as a simple text with tags having the depth of 2 at most, we could choose a much simpler storage strategy including the extreme one – a CLOB column.

Another example can be the crucial feature of database storage strategies – updatability of the data. On one hand, we could know that the data will not be updated too much or at all but we need an effective query evaluation. On the other hand, there could be a strong demand for effective data updates, whereas the queries are of marginal importance. And there are of course cases which require effective processing of both. Naturally, the appropriate storage strategies differ strongly. In case of effective query processing a number of indices and numbering schemes can be exploited but at the cost of corresponding expensive updates. Effective updates, conversely, require the simplest information of mutual data relationships. And if both the aspects are required, it is unavoidable to compromise. And such decision can be again made correctly only if we have an appropriate information on the required future usage.

Last but not least, let us consider the question of data redundancy. Without any additional information the optimal storage strategy is the 4NF schema decomposition into relations [4] which can be achieved, e.g., using the Hybrid algorithm [25], a representative of fixed mapping methods. The decomposition does not involve data redundancy or violation of any normal form, i.e. it results in a database schema with the lowest number of relations and null attributes. But, similarly to database design, there can be reasonable real-world cases when the data should not strictly follow the rules of normal forms and their moderation can lead to more effective query processing.

Both the cost-driven and user-driven methods are based on the idea of exploiting additional user-given information and they appropriately adapt the target database schema. In the former case it is extracted from a sample set of XML documents and/or XML queries which characterize the typical future usage, in the latter case it is specified by user-given annotations, i.e. the user directly specifies the required changes of the default mapping. But although there is a plenty of existing representatives of the two approaches, there are still numerous weak points and open issues that should be improved and solved.

The first improvement is searching for identical or similar fragments in the not annotated schema parts. This approach has two main advantages:

1. The user is not forced to annotate all schema fragments that have to be stored alternatively, but only those with different structure. Thus the system is not endangered of unintended omitting of annotating all similar cases.

2. The system can reveal structural similarities which are not evident "at first glance" and which could remain hidden to the user.

Thus the first main concern of the proposal is how to identify identical or similar fragments within the schema.

The second enhancing focuses on the choice of the mapping strategy for schema fragments which were neither annotated by the user, nor identified as fragments similar to the annotated ones. In this case we combine the idea of cost-driven methods with the fact that a user-driven technique should support various storage strategies too. Hence the second concern is how to find the optimal mapping strategy for the remaining schema fragments and, in addition, with exploitation of the information we already have, i.e. the user-specified annotations, as much as possible.

3 Related Work

As we have mentioned, methods which involve a user in the mapping process can be divided into user-defined and user-driven. Probably due to simple implementation the former ones are supported in most commercial database systems [2]. On the other hand, the set of techniques of the latter type is surprisingly small. To our knowledge there are just two main representatives of the approach – so-called *Mapping Definition Framework (MDF)* [11] and *XCacheDB System* [4]. Both support inlining and outlining of an element / attribute, mapping an element / attribute to a CLOB column, renaming target tables / columns, and redefining column data types. The former approach furthermore supports the *Edge mapping* [12] strategy and enables to specify the required capturing of the structure of the whole schema. The latter one, in addition, allows a certain degree of redundancy.

In both the cases the mapping for not annotated parts is fixed and the annotations are applied just directly on the annotated schema fragments. The two ideas we want to use for their enhancing are adaptivity [19] and similarity [20].

3.1 Adaptive XML-to-Relational Mapping

Probably the first proposal of an adaptive cost-driven method can be found in [16]. It is based on the idea of storing well structured parts of XML documents into relations (using the 4NF decomposition) and semi-structured parts using an *XML data type*, which supports path queries and XML-aware full-text operations. The main concern of the method is to identify the structured and semi-structured parts. For this purpose a sample set of XML documents and XML queries is used.

The other existing cost-driven approaches [8] [28] [30] use a different strategy. They define a set of XML-to-XML transformations (e.g. inlining / outlining of an element / attribute, splitting / merging of a shared element[2], associativity,

[2] An element with multiple parent elements in the schema – see [25].

commutativity, etc.), a fixed XML-to-relational mapping, and a cost function which evaluates a relational schema against a given sample set of XML data and/or queries. Using a search algorithm a space of possible relational schemes is searched and the optimal one is selected. Since it can be proven that even a simple set of transformations causes the problem to be NP-hard, the corresponding search algorithms in fact search for suboptimal solutions and exploit, e.g., heuristics, terminal conditions, approximations, etc.

3.2 Similarity of XML Data

Exploitation of similarity of XML data can be found in various XML technologies, such as, e.g., document validation, query processing, data transformation, storage strategies based on clustering, data integration systems, dissemination-based applications, etc. Consequently, the number of existing works is enormous. We can search for similarity among XML documents, XML schemes, or between the two groups. Furthermore, we can distinguish several levels of similarity that can be taken into account during the search process – a structural level (i.e. considering only the structure of the given XML fragments), a semantic level (i.e. taking into account also the meaning of element / attribute names), a constraint level (i.e. taking into account also various text value constraints), etc.

In case of document similarity we distinguish techniques expressing the similarity of two documents D_1 and D_2 by measuring how difficult is to transform D_1 into D_2 or vice versa (e.g. [23]) and techniques which specify a simple and reasonable representation of D_1 and D_2 that enables their efficient comparison and similarity evaluation (e.g. [29]). In case of similarity of document D and schema S there are also two types of strategies – techniques which measure the number of elements which appear in D but not in S and vice versa (e.g. [6]) and techniques which measure the closest distance between D and all documents valid against S (e.g. [22]). Finally, methods for measuring similarity of two XML schemes S_1 and S_2 exploit and combine various supplemental information and measures such as, e.g., predefined similarity rules, similarity of element / attribute names, equivalence of data types and structure, schema instances, thesauri, previous results, etc. (e.g. [17] [10] [26])

4 Proposed Algorithm

A general idea of fixed schema-driven XML-to-relational mapping methods is to *decompose* the given XML schema S into a set of relations $R = \{r_1, r_2, ..., r_n\}$ using a mapping strategy s_{rel}. An extreme case is when S is decomposed into a single relation resulting in many null values. Other extreme occurs when for each element $e \in S$ a single relation is created resulting in numerous join operations. (Note that since fixed mapping methods view an XML document as general directed tree with several types of nodes, we can speak about schema decomposition too.)

In user-driven strategies the decomposition is influenced by user-defined *annotations* which specify how a particular user wants to store selected schema fragments $F = \{f_1, f_2, ..., f_m\}$. The user usually provides S (i.e. selected elements determining the fragments) with *annotating attributes* from the predefined set of attribute names Ω_A, each of which represents a particular fixed mapping strategy, resulting in an *annotated schema* S'. A classical user-driven strategy then consist of the following steps:

1. S is annotated using Ω_A resulting in S'.
2. Annotated fragments from F are decomposed according to appropriate mapping methods.
3. Not annotated fragments of S' are decomposed using a default fixed mapping strategy s_{def}.

The method enhances a classical user-driven strategy combining it with the idea of adaptive approaches. We simply add the following steps between the step 1 and 2:

a. For $\forall\ f \in F$ we identify a set F_f of all fragments similar to f occurring in $S \backslash \{f\}$.
b. For $\forall\ f \in F$ all fragments in F_f are annotated with annotating attributes of f.
c. $S \backslash F$ is annotated using an adaptive strategy.

The whole mapping process is schematically depicted in Figure 1 where the given schema S with $F = \{f, g\}$ is mapped to a database schema R. If the proposed enhancing, i.e. steps 1.a – 1.c, are included, the system gradually identifies and adds new annotated fragments f_1, f_2, g_1, g_2, and g_3 which are mapped using user-required mapping strategies. If the enhancing is not included (i.e. in case of a classical user-driven strategy), only fragments f and g are annotated using user-required strategies and the rest of the schema using s_{def}.

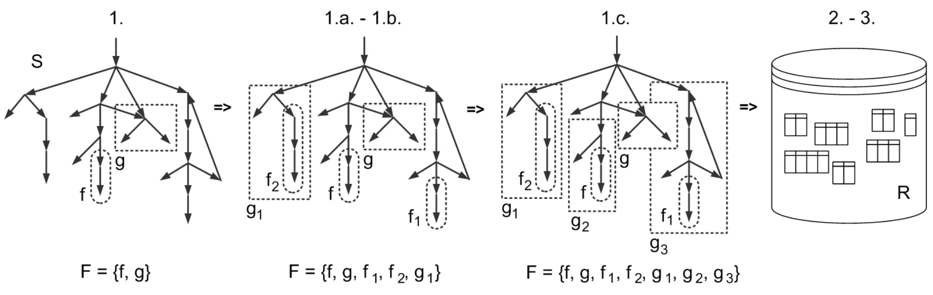

Fig. 1. Schema of the mapping process

As it is obvious, the basic ideas are relatively simple. But if we analyze the strategies more deeply, several interesting issues and open problems that need to be solved occur. They are described in the following chapters.

4.1 Searching for Similar Fragments

As we have mentioned, there are numerous approaches to measuring similarity of XML data. Nevertheless, most of them cannot be directly used for our case since our demanded key characteristics differ. In particular, we search for similarity within the scope of a single schema, the similarity measure should not depend on similarity of element / attribute names but primarily on complexity of content models, and the similarity measure cannot obviously depend on the context of fragments.

Considering the problem in more depth, several fundamental questions arise:

1. How are the annotated fragments defined?
2. What types of annotations, i.e. fixed mapping strategies, are supported?
3. What measure is used for measuring similarity of two schema fragments?
4. Can we optimize the exhaustive search strategy?

The answers for the questions mutually influence each other and specify the algorithm. Furthermore, the definition of annotated fragments together with the question of their mutual intersection are closely related to supported mapping strategies.

Annotated Fragments First of all, for easier processing we define a graph representation of an XML schema S, no matter if annotated or not. For easier explanation we assume that the given XML schema S is expressed in DTD[3] [9], nevertheless the algorithm can be applied to schemes expressed using, e.g., XML Schema [27] [7] language as well.

Definition 1. *A* schema graph *of an XML schema S is a directed, labeled graph $G_S = (V, E, \Sigma_E, \Sigma_A, lab)$, where*

- *V is a finite set of nodes,*
- *$E \subseteq V \times V$ is a set of edges,*
- *Σ_E is a set of element names in S,*
- *Σ_A is a set of attribute names in S, and*
- *$lab : V \to \Sigma_E \cup \Sigma_A \cup \{$"|", "*", "+", "?", ","$\} \cup \{pcdata\}$ is a surjective function which assigns a label to $\forall v \in V$.*

Definition 2. *A* fragment *f_e of a schema S is each subgraph of G_S consisting of an element e, all nodes reachable from e, and corresponding edges. A node f is* reachable *from e if there exists a directed path from e to f in G_S.*

Φ is a set of all element fragments of S.

Next, we assume that each annotated fragment $f \in F$ is uniquely determined by the element e which was annotated using an annotating attribute $a \in \Omega_A$.

Definition 3. *An* annotated element *e of schema S is an element provided with an annotated attribute from Ω_A.*

[3] We omit supplemental constructs such as entities, CDATA sections, comments, etc.

Definition 4. *An* annotated fragment f_e *of schema* S *is a fragment of* S *rooted at an annotated element* e *excluding all annotating attributes from* Ω_A.

As we want to support shared elements and recursion, since both the constructs are widely used in real XML data [21], we must naturally allow the annotated fragments to intersect almost arbitrarily. To simplify the situation, we define an *expanded schema graph* which exploits the idea that both the constructs purely indicate repeated occurrence of a particular pattern.

Definition 5. *An* expanded schema graph G_S^{ex} *is a result of the following transformations of schema graph* G_S*:*

1. *Each shared element is duplicated for each sharer using a* deep copy operation*, i.e. including all its descendants and corresponding edges.*
2. *Each recursive element is duplicated for each repeated occurrence using a* shallow copy operation*, i.e. only the element node itself is duplicated.*

An illustrative example of a schema graph G_S and its expanded schema graph G_S^{ex} is depicted in Figure 2. A shared element is highlighted using a dotted rectangle, a recursive element is highlighted using a dotted circle.

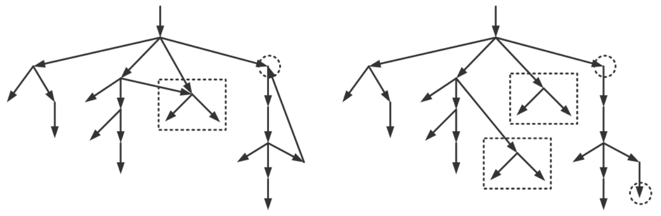

Fig. 2. A schema graph G_S and an expanded schema graph G_S^{ex}

As it is obvious, in case of shared elements the expansion is lossless operation. It simply omits the key advantage of shared elements which allows reusing of previously defined schema fragments. In addition, the real implementation usually does not have to perform the duplication of the shared fragments in fact. The situation is more complicated in case of recursive elements which need to be treated in a special way henceforth. For this purpose we exploit results of statistical analysis of real-world recursive elements [21]. We discuss the details later in the text.

In the following text we assume that a schema graph of an XML schema is always an expanded schema graph, if not explicitly stated alternatively.

Types of Annotations From Definitions 4 and 5 we can easily prove the following two statements:

Lemma 1. *Each expanded schema graph* G_S^{ex} *is a tree.*

Lemma 2. *Two annotated fragments f_x and f_y of an expanded schema graph G_S^{ex} can intersect only if $f_x \subseteq f_y$ or $f_y \subseteq f_x$.*

Furthermore, we can observe that the *common schema fragment*, i.e. the intersection, contains all descendants of a particular element.

We distinguish three types of the annotation intersection depending on the way the corresponding mapping strategies influence each other on the common schema fragment.

Definition 6. *Intersecting annotations are* redundant *if the corresponding mapping strategies are applied on the common schema fragment separately.*

Definition 7. *Intersecting annotations are* overriding *if only one of the corresponding mapping strategies is applied on the common schema fragment.*

Definition 8. *Intersecting annotations are* influencing *if the corresponding mapping strategies are combined resulting in one composite storage strategy applied on the common schema fragment.*

Redundant annotations can be exploited, e.g., when a user wants to store XHTML fragments both in a single CLOB column (for fast retrieval of the whole fragment) and, at the same time, into a set of tables (to enable querying particular items). An example of overriding annotations can occur when a user specifies a general mapping strategy for the whole schema S and then annotates fragments which should be stored differently. Naturally, in this case the strategy which is applied on the common schema fragment is always the one specified for its root element. The last mentioned type of annotations can be used in a situation when a user specifies, e.g., the 4NF decomposition for a particular schema fragment and, at the same time, an additional numbering schema which speeds up processing of particular types of queries. In this case the numbering schema is regarded as a supplemental index over the data stored in relations of 4NF decomposition, i.e. the data are not stored redundantly as in the first case.

Each subset of supported annotations is assigned a (user-specified) intersection type for particular orders of their compositions. This can involve plenty of specifications, but, in fact, the amount of reasonable and thus necessary specifications is much lower than the theoretically allowed ones.

Note that the existing systems [11] [4] mostly support overriding annotations, the XCacheDB system [4], in addition, supports a kind of redundant intersection similar to the above described example.

Search Algorithm The similarity measure, the search algorithm, and its possible optimization are closely related. However, the main idea of the enhancing of user-driven techniques remains the same regardless the chosen measure and algorithm. The choice of the measure influences the precision and scalability of the system, whereas the algorithm influences the efficiency of finding the required fragments.

Let us suppose that we have a similarity measure $sim(f_x, f_y) \in [0, 1]$ expressing similarity of two fragments f_x and f_y of an expanded graph G_S^{ex}, where 1 represents strong similarity and 0 strong dissimilarity, and a similarity threshold $T_{sim} \in [0, 1]$. A naive strategy would exploit an exhaustive search as depicted by Algorithm 1.

Algorithm 1 Single Annotation Strategy (SAS)

```
Input: $S$, $F$, $sim(f_x, f_y)$, $T_{sim}$
Output: $F \cup$ newly annotated fragments
    {construction of the similarity matrix}
 1: $F' \leftarrow F$
 2: for all $f \in F$ do
 3:   for all element $e \in S$ do
 4:     $f_e \leftarrow$ subgraph rooted at $e$
 5:     if $e \in S \backslash \{f\}$ then
 6:       $sim[f, f_e] \leftarrow sim(f, f_e)$
 7:     else
 8:       $sim[f, f_e] \leftarrow 0$
 9:     end if
10:   end for
11: end for
    {annotation strategy}
12: for all element $e \in S$ do
13:   $max_e \leftarrow max_{f \in F}\{sim[f, f_e]\}$
14:   $f_{max} \leftarrow f \in F$ s.t. $sim[f, f_e] = max_e$
15:   if $max_e > T_{sim}$ then
16:     $f_e$.annotation $\leftarrow f_{max}$.annotation
17:     $F' \leftarrow F' \cup \{f_e\}$
18:   end if
19: end for
20: return $F'$
```

The algorithm evaluates similarity of each annotated fragment $f \in F$ and each fragment f_e rooted at an element $e \in S \backslash \{f\}$. We can assume that if the storage strategy for any f_e should not change, the user would mark it as *final*. For such fragment either the default strategy s_{def} or the strategy specified by corresponding user-defined annotation will be used, regardless the results of the search or adaptive algorithm. As such this situation is rather the problem of implementation than a theoretical one and thus we further assume that there are no such fragments in S. On the other hand, we naturally regard fragments annotated by a user to be final by default.

The resulting similarity values are stored into so-called *similarity matrix* $\{sim[f, f_e]\}_{f \in F,\ e \in S}$. An element e is annotated if there exists a fragment $f_{max} \in F$ with the highest similarity value $sim(f_{max}, f_e) > T_{sim}$. In Algorithm 1 we assume that there is always one such candidate at most. Otherwise, the system can ask for user intervention when necessary. We call this approach a *single annotation strategy (SAS).*

The question is whether this approach is correct actually. From another point of view it could be more reasonable to annotate an element e using annotations of all fragments $f \in F$ s.t. $sim(f, f_e) > T_{sim}$. Together with the assumption that for each pair of annotations the result of their composition is predefined and that the annotations have priorities according to which they are composed, this approach seems to be a better choice since it does not omit important information. But, on the other hand, let us consider the situation depicted in Figure 3, where for $i \in \{1, 2, 3\}$ $sim(f, f_i) > T_{sim}$ and f is the annotated fragment.

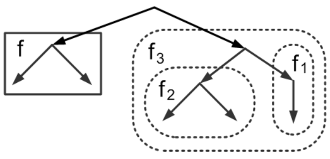

Fig. 3. Similar fragments on the same root path

The problem is whether we can annotate all the three fragments f_1, f_2, f_3 using the annotation of f, especially what will be the result of intersection in case of f_1 and f_3 or f_2 and f_3, i.e. fragments occurring on the same *root path*[4]. We can naturally assume that intersection of two identical annotations is overriding and as such has no effect. Thus we could annotate only the topmost fragment on each root path. In case of example in Figure 3 this rule would be applied twice, resulting in a single annotation of fragment f_3. But what if we knew, in addition, that $sim(f, f_1) > sim(f, f_3)$ and $sim(f, f_2) > sim(f, f_3)$? As it is obvious, in such case it is seems to be more reasonable and natural to annotate fragments f_1 and f_2 rather than whole f_3. Or this situation can be again a case for user intervention, depending on the point of view of it. We will further consider the former one.

If we generalize the idea, the algorithm annotates an element e using annotations of all fragments $f \in F$ s.t. $sim(f, f_e) > T_{sim}$ and $\nexists$ element e' on any root path traversing e s.t. $sim(f, f_{e'}) > sim(f, f_e)$. The resulting algorithm, so-called *multiple annotation strategy (MAS)*, is depicted by Algorithm 2, where *e.ancestors* denotes a set of (direct or undirect) ancestors of element e and *e.descendants* denotes a set of (direct or undirect) descendants of e. The process of construction of the similarity matrix remains the same as in case of Algorithm 1.

Using this approach we should consider what will happen in case a user annotates two structurally identical (or too similar) fragments using different annotations. We cannot simply rely on predefined type of their intersection and corresponding priorities, because the situation is a slightly different one. In this

[4] A path from the root node to a leaf node.

Algorithm 2 Multiple Annotation Strategy (MAS)

```
Input: $S$, $F$, $sim(f_x, f_y)$, $T_{sim}$
Output: $F \cup$ newly annotated fragments
 1: $F' \leftarrow F$
    {construction of the similarity matrix}
 2: -//-
    {annotation strategy}
 3: for all $f \in F$ do
 4:   for all element $e \in S$ do
 5:     if $(sim[f, f_e] > T_{sim}) \wedge$
        $(\nexists\ e_a \in e.ancestors : sim[f, f_{e_a}] > sim[f, f_e]) \wedge$
        $(\nexists\ e_d \in e.descendants : sim[f, f_{e_d}] > sim[f, f_e])$
         then
 6:       $f_e$.annotation $\leftarrow f$.annotation
 7:       $F' \leftarrow F' \cup \{f_e\}$
 8:     end if
 9:   end for
10: end for
11: return $F'$
```

case the system should rather ask for user intervention whenever it is not able to decide. And this is again a problem of the particular implementation.

Similarity Measure and Optimization of the Search Algorithm Now let us consider the search strategy from the point of view of complexity of the algorithm. Figure 4 depicts an example of processing a single annotated fragment, in particular the amount of similarity comparisons. Annotated fragments f and g are highlighted using rectangles, all schema fragments, which are compared with f are highlighted using dotted ovals.

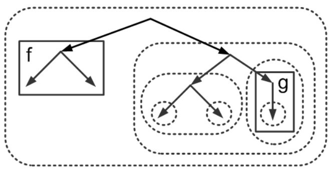

Fig. 4. Exhaustive search strategy

If we do not know any features of the measure, there are not many ways how to avoid the exhaustive search. Also the order in which fragments in F are processed is then unimportant. But although we can assume that $card(F) = m$ is small, i.e. that a user annotates several fragments but the number is not large, the exhaustive search can be expensive due to the size of G_S^{ex}. And even from the simple example in Figure 4 it is obvious that there are pairs of schema fragments which do not have to be compared at all. Another problem is the complexity of the if condition of Algorithm 2 (line 5) which can in the worst case lead to multiple searching through the whole G_S^{ex}. So in both the cases we need to avoid the unnecessary similarity evaluations.

It seems promising to borrow the idea of clustering, similarly to paper [26], where the distance between schema fragments is determined by their mutual similarity, e.g. $dist(f_x, f_y) = 1 - sim(f_x, f_y)$. An example is depicted in Figure 5 for the sample schema in Figure 4.

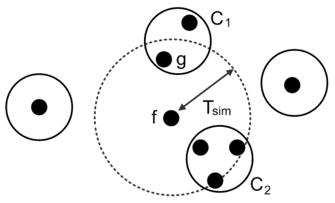

Fig. 5. Exploitation of clustering

All schema fragments (depicted using black-filled circles) are divided into clusters C_1, C_2,..., C_k (depicted using black circular lines) having their centroids c_1, c_2,..., c_k and radii r_1, r_2,..., r_k (or one common radius r, depending on the implementation). Having this information, only those schema fragments have to be compared with fragment f, whose clusters intersect the cluster with centroid f and radius T_{sim}. In case of Figure 5 these are clusters C_1 and C_2. Obviously, if the clusters were selected appropriately, the amount of comparisons would decrease rapidly. Hence the key concern of all clustering algorithms is mainly the construction of the clusters.

The construction is usually performed using a *k-means algorithm* or its variations (e.g. [26]), where the initial clusters are selected randomly and then iteratively improved. In the i-th iteration each fragment is compared with centroids of all clusters and assigned to the closest one. The algorithm terminates if none of the clusters changes, otherwise new centroids are computed and $(i + 1)$-th iteration follows. The complexity of the construction is $O(I \cdot |\Phi| \cdot k)$, where I is the number of iterations. In case of complexity of similarity evaluation the worst case is when either $k = 1$ or all k clusters mutually intersect, i.e. when we cannot avoid any of the similarity comparisons. Hence in the worst case the number of comparisons is the same as in the exhaustive search strategy and the complexity can worsen only the pre-processing, i.e. the construction of clusters. And this is the step we want to remove too.

For further optimization we can exploit characteristics of the chosen similarity measure. The existing algorithms for measuring similarity on schema level usually exploit various supplemental *matchers* [24], i.e. functions which evaluate similarity of a particular feature of the given schema fragments, such as, e.g., similarity of leaf nodes, similarity of root element names, similarity of context, etc.

Definition 9. *A* matcher *is a function $m : \Phi^2 \rightarrow [0,1]$ which evaluates similarity of a particular feature of two schema fragments $f_x, f_y \in \Phi$.*

Definition 10. *A* partial similarity measure *is a function $m_{part} : \Phi^2 \rightarrow [0,1]^p$ which evaluates similarity of the given schema fragments $f_x, f_y \in \Phi$ using matchers $m_1, m_2, ..., m_p : \Phi^2 \rightarrow [0,1]$ and returns a p-tuple of their results.*

Then the partial results are combined using an appropriate approach into the resulting composite similarity value. The most common and verified one [10] is a kind of a weighted sum.

Definition 11. *A* composite similarity measure *is a function $m_{comp} : [0,1]^p \rightarrow [0,1]$ which aggregates the results of particular matchers and returns the total similarity value.*

We can also state the following axioms:

Axiom 1 $m(f, f) = 1; \forall f \in \Phi$

Axiom 2 $m(f, f') = m(f', f); \forall f, f' \in \Phi$

Axiom 3 $m_{part}(f, f) = [1, ..., 1]; \forall f \in \Phi$

Axiom 4 $m_{part}(f, f') = m_{part}(f, f'); \forall f, f' \in \Phi$

For most of the usually used matchers (such as e.g. similarity of number of elements, similarity of depths, etc.) the knowledge of actual value of the analyzed feature for child nodes is necessary for evaluating the value for their parent node. Thus the existing techniques usually exploit a bottom-up strategy, i.e. starting from leaf nodes towards the root node, and search for schema fragments exceeding the threshold T_{sim}. Together with the previously mentioned problem of similar intersecting fragments this is why we need to know the behavior of the similarity measure on particular root paths.

For instance, if we knew that the similarity measure is concave, i.e. that it has only one global maximum, we could skip processing of all the ancestors on the current root path whenever we reach the fragment with the extreme value. A sample situation can be seen in Figure 6 which depicts an example of a graph of similarity function for an annotated fragment f and fragments $f_1, f_2, ..., f_r$ on a single root path. From the graph we can see, that only fragments f_1, f_2, f_3, f_4 need to be processed (f_4 for testing the extremity), then the similarity evaluation can terminate, skipping fragments $f_5, f_6, ..., f_r$.

As it is obvious, this way we can decrease the number of unnecessary similarity evaluations as well as avoid pre-processing of the schema and expensive checking of the if condition of Algorithm 2. Naturally, the efficiency of such approach depends strongly on the position of the extreme on the root path. The

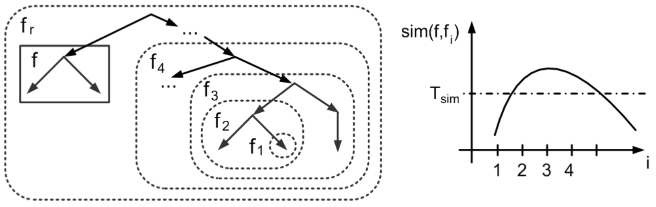

Fig. 6. Exploitation of behavior of similarity function

key problem is how to define such similarity measure. For our purpose we need a measure which focuses especially on the structure of the compared fragments, known equivalences or relations between regular expressions, differences between simple and complex types, etc., i.e. on features that influence the efficiency of database processing the most. But it is hard, if not impossible, to propose a measure with concave behavior which is at the same time enough precise in relation to these requests. Nevertheless, we can exploit a relaxed version of this idea as a kind of heuristics of the bottom-up strategy.

Although we can hardly ensure that m_{comp} is concave, we can assume that at least q of the matchers, where $1 \leq q \leq p$, have this property. For instance a trivial matcher with such behavior can compare the number of distinct element or attribute names, the number of similar operators, the depth of the corresponding content model, etc. Such information is then used as a heuristics based on the idea that if at least "sufficient amount" of the q matchers exceed their extreme value, we can terminate processing of the current root path too.

The whole optimization of the approach, so-called *basic annotation strategy (BAS)*, is depicted by Algorithm 3, where function *terminate* returns *true* if the search algorithm should terminate in the given node, otherwise it returns *false*. Furthermore, we assume that each element of the graph is assigned an auxiliary list of *candidates* consisting of pairs $\langle fragment, similarity \rangle$, i.e. references to fragments (and corresponding similarity values) within its subtree that are candidates for annotation.

The algorithm processes schema graph starting from leaf nodes. For each root path the optimal similarity value and the reference to corresponding fragment are propagated until a better candidate is found or the condition of the heuristics is fulfilled. Then the processing of the current root path is terminated and current candidates are annotated. The complexity of the algorithm depends on the heuristics. In the worst case it does not enable to skip processing of any node that results in the exhaustive search.

In general we could use an arbitrary similarity measure, not exactly the above defined composite one. It is also possible to use disjoint sets of matchers for the heuristics and for the composite similarity measure. Nevertheless, we deal with the above described ones, since it is the typical and verified way for evaluating similarity among XML schemes.

Algorithm 3 Basic Annotation Strategy (BAS)

```
Input: $S$, $F$, $m_1$, $m_2$, ..., $m_q$, $m_{q+1}$, ..., $m_p$, $m_{comp}$, $T_{sim}$
Output: $F \cup$ newly annotated fragments
 1: $F' \leftarrow F$
 2: for all $f \in F$ do
 3:     listToProcess $\leftarrow$ leaf elements of $G_S^{ex} \backslash \{f\}$
 4:     listOfProcessed $\leftarrow \emptyset$
 5:     while listToProcess $\neq \emptyset$ do
 6:         for all $e \in$ listToProcess do
 7:             $e$.candidates $\leftarrow \emptyset$
 8:             $f_e \leftarrow$ subgraph rooted at $e$
 9:             $sim_e \leftarrow m_{comp}(f, f_e)$
10:             for all $c \in e$.subelems do
11:                 for all $\langle f', sim \rangle \in c$.candidates do
12:                     if $sim > sim_e$ then
13:                         $e$.candidates $\leftarrow e$.candidates $\cup \{\langle f', sim \rangle\}$
14:                     end if
15:                 end for
16:             end for
17:             if $e$.candidates $= \emptyset \wedge sim_e > T_{sim}$ then
18:                 $e$.candidates $\leftarrow e$.candidates $\cup \{\langle f_e, sim_e \rangle\}$
19:             end if
20:             if terminate($f$, $e$, $m_1$, $m_2$, ..., $m_q$, $T_{sim}$) then
21:                 for all $\langle f', sim \rangle \in e$.candidates do
22:                     $f'$.annotation $\leftarrow f$.annotation
23:                     $F' \leftarrow F' \cup \{f'\}$
24:                 end for
25:             else
26:                 if $\forall\ s \in e$.siblings : $s \in$ listOfProcessed then
27:                     listToProcess $\leftarrow$ listToProcess $\cup$ {$e$.parent}
28:                 end if
29:             end if
30:             listToProcess $\leftarrow$ listToProcess $\backslash \{e\}$
31:             listOfProcessed $\leftarrow$ listOfProcessed $\cup \{e\}$
32:         end for
33:     end while
34: end for
35: return $F'$
```

Recursive Elements Last but not least, we have to solve the open problem of expanded recursive elements, since the expansion is not a lossless operation as in case of shared elements. We exploit the results of analysis of real-world XML data [21] which shows two important aspects:

1. Despite it is generally believed that recursive elements are of marginal importance, they are used in a significant portion of real XML data.
2. Although the recursive elements can have arbitrarily complex structure, the most common type of recursion is linear and the average depth of recursion is low.

If we realize that we need the "lost" information about recursion only at one stage of the algorithm, the solution is quite obvious. We analyze the structure of schema fragments when evaluating matchers $m_1, m_2, ..., m_p$, whereas each of the matchers describes similarity of a particular feature of the given fragments. In case the fragments contain recursive elements we will not use the exact measure, but its approximation with regard to the real complexity of recursive elements. For instance if the matcher analyzes the maximum depth of fragment containing a recursive element, the resulting depth is not infinite, but considers the average depth of real-world recursive elements.

The question is whether it is necessary to involve a matcher which analyzes the amount of recursive elements in schema fragments. On one hand, it can increase the precision of the composite measure. But from another point of view the approximation transforms the recursive element to a "classical" element and hence such matcher can be misleading.

4.2 Adaptive Mapping Strategy

At this stage of the algorithm we have a schema S and a set of annotated fragments F which involve the user-defined fragments and fragments identified by BAS algorithm. As the second enhancing we apply an adaptive mapping strategy on the remaining parts of the schema. At first glance the user-driven techniques have nothing in common with the adaptive ones. But under a closer investigation we can see that the user-given annotations provide a similar information – they "say" how particular schema fragments should be stored to enable efficient data querying and processing. Thus we can reuse the user-given information. For this purpose we define an operation *contraction* which enables to omit those schema fragments where we already know the storage strategy and focus on the remaining ones.

Definition 12. *A* contraction *of a schema graph G_S with annotated fragment set F is an operation which replaces each fragment $f \in F$ with a single auxiliary node called a* contracted node. *The resulting graph is called a* contracted graph *G_S^{con}.*

The basic idea of the adaptive strategy is as follows: Having a contracted graph G_S^{con} we repeat the BAS algorithm and operation contraction until there is no fragment to annotate. The BAS algorithm is just slightly modified:

– It searches for schema fragments which are not involved in the schema, i.e. it searches among all nodes of the given graph and returns the (eventually empty) set of found fragments.
– For similarity evaluation we naturally do not take into account contracted nodes.
– The annotations of contracted nodes are always overriding in relation to the newly defined ones.

We denote this modification of BAS as a *contraction-aware annotation strategy (CAS)*. The resulting annotating strategy, so called *global annotation strategy (GAS)*, is depicted by Algorithm 4, where function *contract* applies operation contraction on graph of the given schema S and set of fragments F and function *expand* expands all the contracted nodes of the given schema to the original ones.

Algorithm 4 Global Annotation Strategy (GAS)

```
Input: $S$, $F$, $m_1$, $m_2$, ..., $m_p$, $m_{comp}$, $T_{sim}$
Output: $F \cup$ newly annotated fragments
1: $F' \leftarrow$ BAS($S$, $F$, $m_1$, $m_2$, ..., $m_p$, $m_{comp}$, $T_{sim}$)
2: $F^{tmp} \leftarrow F'$
3: while $F^{tmp} \neq \emptyset$ do
4:     contract($S$, $F^{tmp}$)
5:     $F^{tmp} \leftarrow$ CAS($S$, $F$, $m_1$, $m_2$, ..., $m_p$, $m_{comp}$, $T_{sim}$)
6:     $F' \leftarrow F' \cup F^{tmp}$
7: end while
8: expand($S$, $F'$)
9: return $F'$
```

The resulting complexity of the algorithm depends on the number of iterations of the cycle (lines 3 – 7). In the worst case each iteration results in annotating of a single element, i.e. the search algorithm repeats $(|\Phi| - |F| + 1)$ times.

4.3 Open Issues

Despite the detailed description of the algorithm, possible solutions and their consequences, there are still several open issues. We distinguish two main categories:

1. features of the particular implementation and
2. behavior of the algorithm on real XML data.

As for the former case the key implementation decisions are especially:

1. matchers m_1, m_2, ..., m_q, m_{q+1}, ..., m_p,

2. the composite similarity measure m_{comp},
3. threshold T_{sim}, and
4. the set of supported schema annotations and types of their mutual intersection (or forbiddance).

The key problem lies especially in tuning of weights of the composite similarity measure, since they influence the precision of the system strongly. The remaining characteristics are related rather to its usefulness and versatility. There are also marginal questions such as, e.g., whether the system will support final elements or user intervention if there are more candidates for a particular situation. But these features do not have the key influence on the proposed approach itself.

The latter category of open issues is quite unpredictable, despite the existing statistics of real XML data [21]. It is caused mainly by two facts. Firstly, although we know the usual characteristics of the real data, we cannot predict especially the behavior of more complex similarity measures due to the above mentioned tuning of the system. And secondly, we cannot predict the behavior of the proposed adaptive strategy, since we have no information about the structure of contracted graphs of real data. Furthermore, the choice of particular schema fragments will be strongly related to the type of the tested data and thus the efficiency of the resulting storage strategy can vary remarkably.

As it is obvious, both the categories are also related significantly. And though some particular features can be estimated or preset according to the existing user-driven systems and statistical analysis of data, most of them still require a series of experimental tests.

5 Experimental Implementation

For testing and evaluation of key features of the proposed algorithm we have implemented an experimental system called *UserMap* and performed various tests on real XML data. In the following sections we first describe the particular implementation decisions and then the corresponding results of the tests.

5.1 Similarity Evaluation

As we have mentioned, the similarity measure used for the algorithm should focus mainly on structural aspects of the given schema fragments and especially constructs which influence the database processing. Thus it should not rely, e.g., on semantic of element or attribute names, as it is usual in existing similarity evaluation algorithms [20]. Furthermore, since the algorithm assumes that a user can annotate any kind of a schema fragment and such fragment can be detected by the search algorithm anywhere in the schema (except for final schema fragments), it should not rely also on context of the evaluated fragments. An last but not least, the similarity evaluation strategies have to cope with the problem of setting various parameters, such as, e.g., weights of particular matchers within the composite measure. The setting process is usually omitted in the descriptions (i.e. the weights are set straightforwardly without any argumentation), or a kind of a machine-learning technique is exploited.

With regard to these observations we have decided to exploit the knowledge and results of statistical analysis of real XML data collections in [21], in particular the part considering XML schemes. The data parameters that were analyzed describe a given schema fragment (in case of the analysis the whole schema) quite precisely and at the same time the results over a representative sample of real XML schemes can be used for tuning parameters of the similarity measure.

Data Characteristics Before we describe the matchers and the composite similarity measure we repeat several definitions of XML data characteristics (or their slight modifications) used in [21] whose knowledge is necessary for understanding the following text.

In most of the existing works DTDs (or XSDs, i.e. XML Schema definitions) are viewed as sets of regular expressions over element names. Attributes are often omitted for simplicity, since they form unordered sets of values having only a simple type. Thus the key characteristic of a schema fragment is a *content model* and its complexity.

Definition 13. A content model α *over a set of element names* Σ'_E *is a regular expression defined as* $\alpha = \epsilon \mid pcdata \mid f \mid (\alpha_1, \alpha_2, ..., \alpha_n) \mid (\alpha_1|\alpha_2|...|\alpha_n) \mid \beta* \mid \beta+ \mid \beta?$, *where* ϵ *denotes* an empty content model, *pcdata denotes* a text content model, $f \in \Sigma'_E$, *"," and "|" stand for concatenation and union (of content models* α_1, α_2, ..., α_n*), and "*", "+", and "?" stand for zero or more, one or more, and optional occurrence(s) (of content model* β*) respectively.*

A content model α *s.t.* $\alpha \neq \epsilon \wedge \alpha \neq pcdata$ *is called* an element content model.

The complexity of a content model can be viewed from various points of view. The basic characteristics simply distinguish *empty*, *text*, and *element* content of an element, more complex ones involve the *depth* and width of a schema fragment (characterized by various types of *fan-out*).

Definition 14. A depth of a content model α *is inductively defined as follows:*

$depth(\epsilon) = 0$;

$depth(pcdata) = depth(f) = 1$;

$depth(\alpha_1, \alpha_2, ..., \alpha_n) = depth(\alpha_1|\alpha_2|...|\alpha_n) = max(depth(\alpha_i)) + 1$, *where* $i = 1, 2, ..., n$;

$depth(\beta*) = depth(\beta+) = depth(\beta?) = depth(\beta) + 1$.

Definition 15. An element fan-out *of an element e is the number of distinct elements in content model* α*, where* $e \rightarrow \alpha$*.*

An attribute fan-out *of an element e is the cardinality of set* β*, where* $e \rightarrow \beta$*.*

Definition 16. A minimum element fan-out *of element e is the minimum number of elements allowed by its content model* α*.*

A maximum element fan-out *of element e is the maximum number of elements allowed by content model α.*

An unbounded element fan-out *is a maximum element fan-out of* ∞*.*

Even more complex characteristics describe special types of content model, their features, and variations, such as trivial, unordered, recursive, or mixed content of an element, or so-called relational and DNA patterns.

Definition 17. *An element is* trivial *if it has an arbitrary amount of attributes and its content model $\alpha = \epsilon \mid pcdata$.*

A mixed-content element is simple *if each of its subelements is trivial. A mixed-content element that is not simple is called* complex.

A simple element fan-out *of an element e is the number of distinct trivial elements in its content model α.*

Definition 18. *An element e is* recursive *if e is reachable from e.*

An element e is trivially recursive *if it is recursive and e is the only element reachable from e and neither of its occurrences is enclosed by "*" or "+".*

An element e is linearly recursive *if it is recursive and e is the only recursive element reachable from e and neither of its occurrences is enclosed by "*" or "+".*

An element e is purely recursive *if it is recursive and e is the only recursive element reachable from e.*

An element that is recursive but not purely recursive is called a generally recursive *element.*

Definition 19. *A content model α is* mixed*, if $\alpha = (\alpha_1|...|\alpha_n|\, pcdata)* \mid (\alpha_1|...|\alpha_n|pcdata)+$ where $n \geq 1$ and for $i = 1, 2, ..., n$ content model $\alpha_i \neq \epsilon \wedge \alpha_i \neq pcdata$.*

An element e is called mixed-content element *if content model α, where $e \rightarrow \alpha$, is mixed.*

Definition 20. *A nonrecursive element e is called* a relational pattern *if it has an arbitrary amount of attributes and its content model $\alpha = (e_1, e_2, ..., e_n)* \mid (e_1, e_2, ..., e_n)+ \mid (e_1|e_2|...|e_n)* \mid (e_1|e_2|...|e_n)+$, where e_1, e_2, ..., e_n are trivial elements.*

A nonrecursive element e is called a shallow relational pattern *if it has an arbitrary amount of attributes and its content model $\alpha = f* \mid f+$, where f is a trivial element.*

Definition 21. *A nonrecursive element e is called* a DNA pattern *if it is not mixed and its content model α consists of a nonzero amount of trivial elements and one nontrivial and nonrecursive element whose occurrence is not enclosed by "*" or "+". The nontrivial subelement is called* a degenerated branch.

A depth *of a DNA pattern e is the maximum depth of its degenerated branch.*

Matchers and Composite Measure For definition of matchers m_1, m_2, ..., m_p we exploit most of the XML data characteristics evaluated in the analysis [21], as defined in previous section. Since we want to describe the structure of the schema fragments as precisely as possible, the amount of characteristics is nontrivial. On the other hand, at this stage the versatility of the approach becomes evident, since in general any kind of matchers can be used depending on the purpose and requirements of corresponding application.

According to the scope the used characteristics can be divided into the following groups:

- *root* – characteristics of root node of the fragment:
  - type of content (empty, text, element, mixed, trivial, or unordered),
  - fan-outs (element, attribute, simple, minimum, and maximum),
  - type of the node (DNA pattern or relational pattern), and
  - type of the recursion (trivial, linear, pure, or general).
- *subtree* – characteristics of the whole fragment:
  - basic (number of elements, number of attributes, number of mixed contents, number of empty contents, maximum and average depth, minimum and maximum element fan-out),
  - XML Schema-like (number of unordered contents, default values, fixed values, wildcards, `ID` types, `IDREF(S)` types, `unique`, `key`, and `keyref` nodes),
  - recursive (number of recursive elements, number of particular types of recursion),
  - depths (minimum, maximum, average, and unbounded),
  - fan-outs (element, simple, minimum, and maximum),
  - mixed contents (depths, fan-outs, simple fan-outs),
  - DNA patterns (depths, fan-outs, simple fan-outs), and
  - relational patterns (fan-outs, simple fan-outs).
- *level* – characteristics of each level of the fragment: number of elements, attributes, text nodes, mixed contents, DNA patterns, and relational patterns, fan-outs and simple fan-outs.

Since each matcher should evaluate similarity of particular characteristic of the given schema fragments f_x and f_y, we need to transform the resulting values (e.g. types of the root nodes, numbers of elements, etc.) to interval $[0, 1]$. In case of root characteristics we distinguish two cases – feature matchers and single-value matchers. *Feature matchers* express the (in)equality of the value of i-th feature fea_i (e.g. type of the node, content, or recursion):

$$m_i^{fea}(f_x, f_y) = \begin{cases} 1 & fea_i(f_x) = fea_i(f_y) \\ 0 & otherwise \end{cases} \tag{1}$$

They are combined using a weighted sum into a *composite feature matcher*:

$$m^{fea}(f_x, f_y) = \sum_{i=1}^{n} m_i^{fea}(f_x, f_y) \cdot w_i^{fea} \tag{2}$$

where weights $w_i^{fea} \in [0,1]$, $\sum_{i=1}^{n} w_i^{fea} = 1$, and n is the number of feature matchers.

Single-value matchers express the difference between the value of j-th single-value characteristic $value_j$ (e.g. element or attribute fan-out):

$$m_j^{single}(f_x, f_y) = \frac{1}{|value_j(f_x) - value_j(f_y)| + 1} \tag{3}$$

As for the subtree characteristics we distinguish two cases too. In case of single-valued characteristics (i.e. basic, XML Schema-like, and recursive) we also use the single-value matchers (3) which are within the subsets composed into *composite single-value matchers* m^{single}, again using a weighted sum. The situation in case of multi-valued characteristics (e.g. list of allowed depths of the fragment) is a bit complicated – it requires similarity evaluation of two lists of values of arbitrary, distinct lengths. Therefore we first supply the shorter list with zero values and sort the lists in decreasing order. Then we use so-called *multi-valued matchers* which express the similarity of a j-th sorted sequence s_j:

$$m_j^{multi}(f_x, f_y) = \frac{\sum_{k=1}^{m} \frac{1}{|s_j(f_x)[k] - s_j(f_y)[k]| + 1}}{m} \tag{4}$$

where m is the length of the sequences and $seq_j(.)[k]$ expresses the k-th member of the sequence.

For level characteristics (e.g. minimum fan-out per level) we use so-called *level matchers* which compose the results of single-valued (3) or multi-valued (4) matchers at particular levels and decrease their weight with the growing level:

$$m_j^{lev}(f_x, f_y) = \sum_{k=1}^{l} m_j^{single/multi}(f_x, f_y) \cdot (\frac{1}{2})^k \tag{5}$$

where l is the maximum of number of levels of f_x and f_y (assuming that the shallower one is again supplied with zero values).

Finally, the resulting composite function m_{comp} is expressed as a weighted sum of all the matchers.

As it is obvious, the resulting composite similarity expresses the similarity of the given schema fragments with regard to the selected matchers, each having its particular weight. Thus there remains the problem of tuning the weights which highly influences the precision of the similarity measure.

Tuning of the System In existing works we can distinguish two approaches – the parameters are set either without any argumentation (or on the basis of authors experience whose more detailed description is usually omitted) or a machine-learning strategy is exploited. In the latter case the corresponding system is usually provided with a set of sample situations (e.g. pairs of data fragments and their similarity) and the system than exploits this knowledge in the evaluation process.

In our approach we use the "golden mean" – we exploit the results from the analysis of real-world XML schemes [21] and we set the parameters on the basis of the results. The basic idea is relatively simple: We use the same 98 real-world XML schemes divided into database (`dat`), document (`doc`), exchange (`ex`), report (`rep`), and research (`res`) category. Their basic characteristics can be seen in Table 1. The first two categories are similar to classical data-centric and document-centric ones, the other three are introduced in [21] to enable finer division.

Characteristic		`dat`	`doc`	`ex`	`rep`	`res`
Number of schemes		31	18	38	4	7
Number of elements	Minimum	7	5	5	109	28
	Maximum	76	377	523	3,213	250
Number of paths	Minimum	5	1	3	97	26
	Maximum	115	11,994	1,665	3,137	568
Depth	Minimum	2	4	2	3	5
	Maximum	12	81	79	5	15

Table 1. Characteristics of XML schemes

We prepare sample patterns of real schema fragments, such as data-centric fragments, document-centric fragments, unordered elements, relational patterns, recursive elements (of all the four types), DNA patterns, etc., whose representation is in the particular categories known. Using a search algorithm we compute the number of occurrences of similar fragments within the schema categories and tune the parameters of the similarity measure so that the results correspond to the results of analysis of the real-world data.

Note that this is the second stage where the algorithm can be modified to any purpose. It general it is possible to use any relevant information, i.e. knowledge of characteristics of any sample set of data. We have used the results of our analysis since the real-world sample is nontrivial and the data were collected so that they cover many possible areas where XML data are exploited.

Theoretical View of the Tuning Problem In general the tuning problem and its proposed solution can be described as follows: Let $c_1, c_2, ..., c_K$ denote the categories of schemes, $p_1, p_2, ..., p_P$ the sample patterns, and $(M^{rep}_{i,j})_{K \times P}$ the *representation matrix* which contains *real-world representation* of pattern p_j in category c_i, i.e. results of the statistics. Next let us have a search algorithm with parameters $par_1, par_2, ..., par_R$, where $\forall i : par_i \in [0, 1]$ and some subsets of the parameters have to fulfill particular constraints, such as, e.g., the sum of subset of parameters which correspond to weights of a single weighted sum must be equal to 1. With the given setting of parameters the algorithm returns *calculated representation* $rep_{i,j}$ of pattern p_j in category c_i. The aim is to find the optimal setting of parameters $par_1, par_2, ..., par_R$, i.e. the setting where the sum of deviations of calculated and real-world representations

$$\Delta = \sum_{i=1}^{K} \sum_{j=1}^{P} |M^{rep}[i,j] - rep_{i,j}| \tag{6}$$

is minimal. This task is obviously a kind of a classical *constraints optimization problem (COP)* [5] – a problem of finding a solution in a *feasible region* (i.e. a set of all possible solutions), where the value of *objective function* (i.e. a function which determines the quality of a solution) is optimal and the solution satisfies the given criteria. In our case:

– the feasible region contains all possible settings of parameters par_1, par_2, ..., par_R corresponding to weights of weighted sums,
– the objective function evaluates Δ, and
– the criteria of the solution are the above described constraints.

The problem is that since the parameters $par_1, par_2, ..., par_R$ are in general real values from $[0,1]$, the feasible region is theoretically infinite.

Under a closer investigation we can see that the real-world case is much simpler than the theoretical problem. Firstly, we can restrict possible values of parameters par_1, par_2, ..., par_R to a certain precision, i.e. the infinite feasible region can be reduced to a reasonable size. Furthermore, for our purpose we do not need the optimal solution, but a reasonably good suboptimum, since the algorithm is expected to search for similar schema fragments, not exactly the given ones. And, last but not least, as the evaluation of the objective function requires similarity evaluation of all the patterns $p_1, p_2, ..., p_P$ and all schema fragments in categories $c_1, c_2, ..., c_K$, we need to minimize the amount of evaluations.

For searching the suboptimum we exploit and compare a slight modification of two approaches for searching a suboptimal solution of an optimization problem – global search heuristics called *genetic algorithms* and *simulated annealing*. They enable to find a reasonable setting following the given requirements and, at the same time, influence the number of expensive evaluations.

Genetic Algorithms *Genetic algorithms (GA)* [13] are a part of evolutionary algorithms which are inspired by observations of evolution biology. Their idea is based on iterative improving of (usually) randomly generated *initial population* P_0 of individuals using two key operations, simulations of natural processes – *crossover* and *mutation*.

As depicted by Algorithm 5, at i-th iteration the *fitness* f_{fit}, i.e. the quality, of every individual of population P_i is evaluated, multiple individuals are selected on the basis of their fitness, and modified, i.e. crossed over and mutated to form a new population P_{i+1}. Operation crossover creates a new offspring crossing over two individuals by exchanging their portions. Operation mutation creates a new individual by changing attributes of an existing one. Both the operations are performed with a given probability P_{cross} and P_{mut} which influence the speed of convergence to the suboptimal solution. The algorithm terminates either if

Algorithm 5 Genetic Algorithm (GA)

```
Input: $f_{fit}(I_x)$, $F_{min}$, $N$
Output: individual with the maximum fitness
 1: $i \leftarrow 0$
 2: $P_i \leftarrow$ initial population
 3: evaluate individuals in $P_i$ using $f_{fit}$
 4: while $i \leq N \wedge F_{min}$ is not reached in $P_i$ do
 5:     $i \leftarrow i + 1$
 6:     $P_i \leftarrow$ best individuals from $P_{i-1}$
 7:     crossover($P_i$)
 8:     mutate($P_i$)
 9:     evaluate individuals in $P_i$ using $f_{fit}$
10: end while
11: return $I \in P_i$ s.t. $f_{fit}(I)$ is maximum
```

satisfactory *fitness level* F_{min} has been reached in population P_i or after N iterations.

In our case a single individual of a population corresponds to a single possible setting of parameters $par_1, par_2, ..., par_R$ and the fitness function evaluates the inverse value of Δ. The initial population is generated randomly or a set of reasonable settings can be used. And finally, operations crossover and mutation are slightly modified to ensure that the subsets of parameters corresponding to a single weighted sum still fulfill the previously described conditions.

Simulated Annealing The idea of *simulated annealing (SA)* [15] is also inspired by natural processes, in this case the way a metal cools and freezes into crystalline structure, where controlled cooling increases size of the crystals and thus reduces defects. SA algorithm is also iterative and based on exploitation of randomly generated solutions.

As depicted by Algorithm 6, the SA algorithm starts with the *initial state* s_0 which is iteratively improved. The quality of a state s_x is evaluated using its *energy* $E(s_x)$ which needs to be minimized. At i-th iteration the current state s_i is replaced with a random "nearby" state s_{i+1} whose choice depends on a global parameter T called *temperature* which is gradually decreased (usually by fixed factor $\alpha < 1$) during the process. The probability P_{mov} of moving from state s_i to s_{i+1} is expressed as a function of T, $E(s_i)$, and $E(s_{i+1})$:

$$P_{mov} = \begin{cases} 1 & E(s_i) > E(s_{i+1}) \\ exp(\frac{E(s_i) - E(s_{i+1})}{T}) & otherwise \end{cases} \tag{7}$$

The algorithm terminates either after a certain number of iterations N or if a state with satisfactory energy E_{min} is reached.

The main advantage of SA in comparison with GA is its ability to avoid trapping at local optimum. The reason is that SA does not accept only states which improve the current optimum, but also some of those which can (temporarily)

Algorithm 6 Simulated Annealing (SA)

```
Input: $E$, $E_{min}$, $N$
Output: state with minimum energy $E$
 1: $i \leftarrow 0$
 2: $s_0 \leftarrow$ initial state
 3: $s_{opt} \leftarrow s_0$
 4: while $i \leq N \wedge E_{min} < E(s_{opt})$ do
 5:     $i \leftarrow i + 1$
 6:     $s_i \leftarrow$ a random neighbor of $s_{i-1}$
 7:     if $E(s_i) < E(s_{opt})$ then
 8:         $s_{opt} \leftarrow s_i$
 9:     end if
10:     if $\neg$ move $(E(s_{i-1}), E(s_i), T)$ then
11:         $s_i \leftarrow s_{i-1}$
12:     end if
13:     decrease($T$)
14: end while
15: return $s_{opt}$
```

worsen it. The probability P_{mov} and temperature T ensure that at the beginning the state changes almost arbitrarily (within the adjacent states) but the changes decrease as T goes to zero.

In our case each state represents a single setting of parameters par_1, par_2, ..., par_R and the energy E evaluates Δ. For the initial state can be again used either a randomly generated setting or any known reasonable setting. The neighboring states are defined by a modification of mutation of GA, where only a single parameter is randomly changed (and the others are recomputed to fulfill conditions of weighted sums).

Experimental Tests For experimental testing of GA and SA algorithms we first need to tune the key parameters of both the algorithms.

As for the GA algorithm, we need to tune the probabilities P_{cross} and P_{mut} which influence the speed of convergence to the suboptimum. With fixed value of P_{mut} and maximum number of iterations $N = 30$ we have performed number of tests for setting the value of P_{cross}. The results are depicted by Figures 7 and 8 containing average, median, and minimum values for values of $P_{cross} \in [0.1, 0.9]$. Figure 7 depicts at which iteration the suboptimal value was reached, Figure 8 depicts the resulting suboptimal value of Δ. As we can see, since both the average and median iteration for reaching the suboptimum occur mostly between 15 and 20, the maximum number of iterations does not need to be much higher. Considering only the reached values (Figure 8) the best candidates for P_{cross} are 0.1, 0.4, 0.5, 0.7 and 0.9. But together with the results from Figure 7 we can say that the optimal value of P_{cross} occurs between 0.4 and 0.5.

Similarly, with fixed value of $P_{cross} = 0.5$ and the same maximum number of iterations $N = 30$ the process of tuning of parameter P_{mut} is depicted in

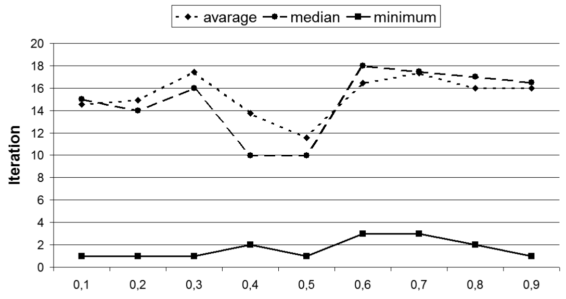

Fig. 7. Tuning of parameter P_{cross} – number of iterations for reaching the suboptimum

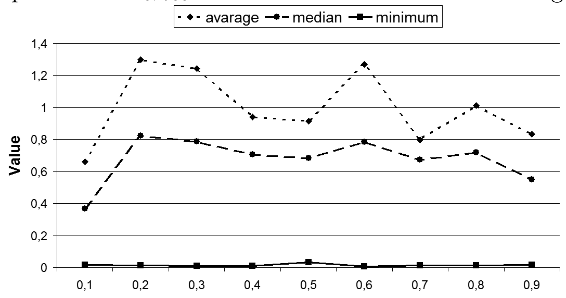

Fig. 8. Tuning of parameter P_{cross} – values of Δ for the suboptimum

Figures 9 and 10 containing values with the same meaning. Unfortunately, these results are more ambiguous than in the previous case. Thus we have analyzed the particular results and from the possible candidates we have selected 0.16 as the best compromise value of P_{mut}.

As for the SA algorithm, we need to perform the same tuning for parameter T and the threshold of P_{mov}. The setting of T requires quite a lot of user involvement since the value must conform to numerous requirements to achieve a reasonable value [15]. Mainly, it should correspond to the estimated deviation of $E(s_i)$ and $E(s_{i+1})$, it must ensure that the algorithm moves to a state s_{i+1} even if $E(s_{i+1}) > E(s_i)$ (but the probability P_{mov} is high enough), the declination of T should be reasonable enough to exploit the main idea of the approach, etc. Thus this parameter was set rather semiautomatically with regard to balanced conformation to the requirements. With fixed value of T and maximum number of iterations $N = 80$ the value of P_{mov} seems to be the best around 0.7 considering the number of iterations and between 0.4 and 0.5 considering the resulting values, as depicted by Figures 11 and 12 respectively. After more detailed analysis of the results, we have set P_{mov} to 0.7.

With the current setting of both the algorithms we can now analyze their behavior. Firstly, we are interested in the quality of the achieved suboptimum, in particular in comparison with the typical reasonable setting of weights so that the composite measure returns the average similarity. Table 2 overviews the quality of the suboptimums expressed using the result of Δ for both the SA

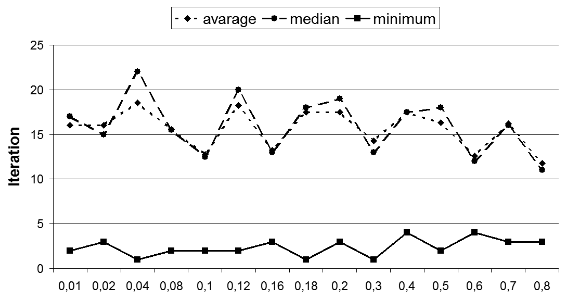

Fig. 9. Tuning of parameter P_{mut} – number of iterations for reaching the suboptimum

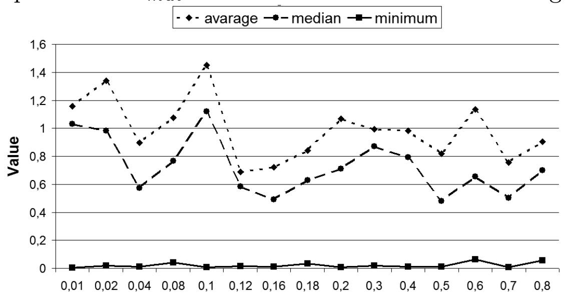

Fig. 10. Tuning of parameter P_{mut} – values of $\varDelta$ for the suboptimum

and GA algorithms which were evaluated either starting with random population P_0/state s_0 or with setting to the average-producing weights (denoted as `avg`). As we can see in both the cases the results are much better when we start with a reasonable, verified setting than with a random one. And, in addition, if we compare the quality of `avg` with the achieved suboptimums, we can see that using both the algorithms can found much better candidates for setting the weights.

Comparing the two algorithms together we are interested in the amount of expensive evaluations of fitness/energy, in particular their efficiency and the quality of the reached suboptimum. As for the quality we can again refer to Table 2, where we can see that though the values of the two algorithms do not differ too much, the GA algorithm performs better in all the cases.

The analysis of number of iterations firstly requires a small discussion: In case of GA the number of evaluations of $\varDelta$ seems to be given by the product of number of iterations and number of individuals in a population. In case of SA it is given by the number of iterations, since at each iteration only a single state is evaluated. But due to the properties of GA all the individuals of a population can be evaluated concurrently avoiding repetitions of expensive preprocessing of the graph and supplemental calculations (e.g. number of nodes, depths, types of nodes, etc.). Thus in fact also in this case the number of evaluations rather corresponds to number of iterations of the algorithm.

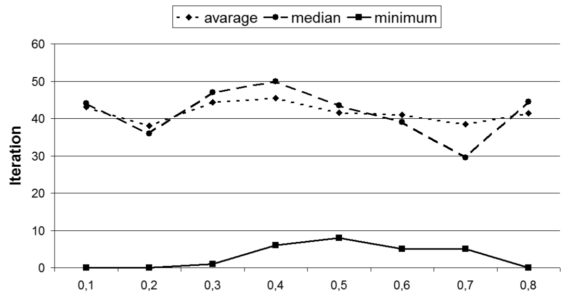

Fig. 11. Tuning of parameter P_{mov} – number of iterations for reaching the suboptimum

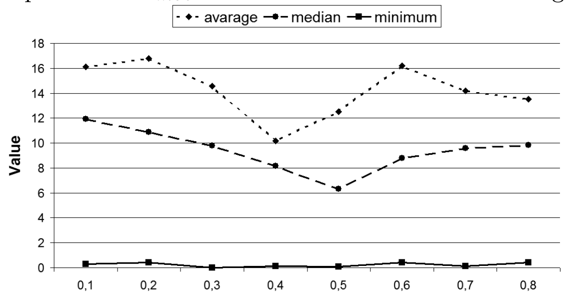

Fig. 12. Tuning of parameter P_{mov} – values of Δ for the suboptimum

The resulting numbers of iterations necessary for reaching the suboptimums from Table 2 are depicted in Table 3. As can be seen not only has the GA algorithm undoubtedly better results than SA, but also the number of iterations is lower due to its ability to evaluate a population of candidates at each iteration instead of a single one. It is quite probable that with a higher number of iterations the SA algorithm would perform better, but the duration of such evaluation would be unacceptable as it cannot be performed concurrently for more candidates. Thus though the SA algorithm is able to cope with local optimums [15], for our purpose seems to be better to use an approach which is able to reach the optimum as soon as possible, as it is in case of GA assured using a population of candidates.

At this stage we have a similarity measure whose parameters are tuned according to the knowledge of structure of real-world data. But the question is how good such tuning is. As we have mentioned the existing works rather focus on semantic similarity of XML schema fragments and thus comparison of our approach with any of them would be misleading. On the other hand, we can compare the tuning with the usually used reasonable setting to the average-producing weights. From Table 2 we can see that in terms of the value of Δ the reached settings are much better than in the average-producing case. But such results are not very convincing in general. Thus for the purpose of evaluation of quality of the two similarity measures (which we further denote as `SimTuned` and `SimAvg`) we use the approach introduced in [10]. It is based on the idea

Characteristic		Result of Δ			
		Minimum	**Average**	**Median**	**Maximum**
P_0 (GA)	Random	0,013	1,176	0,673	3,959
	`avg`	0,001	0,652	0,463	3,441
s_0 (SA)	Random	0,082	17,318	11,764	55,719
	`avg`	0,061	9,412	6,595	40,519
Arithmetic mean		482,113			

Table 2. Quality of the achieved suboptimums

Characteristic		Number of iterations			
		Minimum	**Average**	**Median**	**Maximum**
P_0 (GA)	Random	1	17,2	19	30
	`avg`	5	20,9	22,5	30
s_0 (SA)	Random	8	39,8	38	80
	`avg`	2	38.7	37	80

Table 3. Efficiency of achieving the suboptimum

of comparing results of an algorithm with results of manual processing assuming that the manually achieved results form the optimum. Let R be the set of manually determined matches, i.e. in our case schema fragments similar to the given schema pattern, and P the set of matches determined by the algorithm. Then I denotes the set of *true positives*, i.e. matches correctly identified by the algorithm, $F = P \setminus I$ denotes *false matches*, i.e. matches incorrectly identified by the algorithm, and $M = R \setminus I$ denotes *false negatives*, i.e. matches not identified by the algorithm. On the basis of these characteristics, the following quality measures can be computed:

– *Precision* $= \frac{|I|}{|P|} = \frac{|I|}{|I|+|F|}$ estimates the reliability of the similarity measure,
– *Recall* $= \frac{|I|}{|R|}$ specifies the share of real matches that is found, and
– *Overall* $= 1 - \frac{|F|+|M|}{|R|} = \frac{|I|-|F|}{|R|}$ represents a combined measure which represents the post-match effort necessary to remove false and add missed matches.

In the ideal case $I = P = R$, $F = M = \emptyset$, and the measures reach their highest values Precision = Recall = Overall = 1. On the other hand, the lower the values are (whereas the Overall can have even negative values), the least precise the similarity measure is.

For the purpose of the evaluation we have selected 5 XML schemes representing each of the 5 schema categories and prepared a sample set of 10 data-centric and 10 document-centric schema patterns. For each of the schema representatives we have manually identified the set R of data-centric and document-centric fragments. Then we have performed searching for fragments similar to the schema patterns using both `SimAvg` and `SimTuned`, i.e. we have performed $5 \times (10 + 10)$ experiments for each of them, and determined the sets P and I. Finally, within the categories we have computed average values of the results (i.e. average $|P|$ and $|I|$) and then the resulting values of Precision, Recall, and Overall.

As can be seen from Figure 13 the `SimAvg` approach is apparently much worse similarity measure than `SimTuned` within all the categories. Secondly, it is evident (and natural) that the quality of both the measures is correlated with the number of source schemes within the categories, i.e. the amount of source information we had for tuning the weights. The best results can be found for categories `dat`, `doc`, and `ex` since the amount of corresponding schemes highly exceeds the amount in the other two – see Table 1.

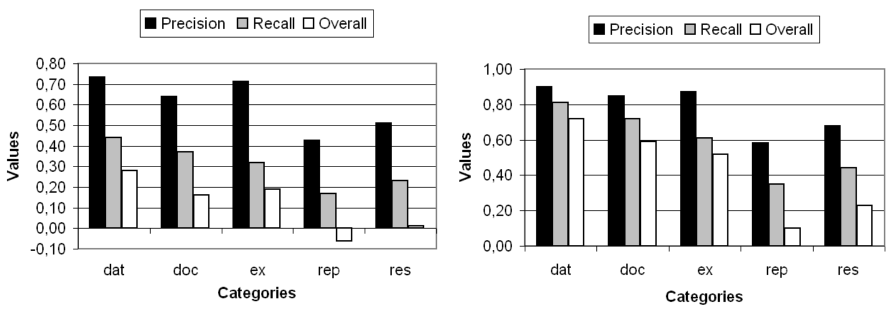

Fig. 13. Precision, Recall, and Overall for `SimAvg` and `SimTuned`

If we consider the Precision and Recall parameters together (since they influence each other and thus cannot be evaluated separately, otherwise the results can be misleading [10]), we can see that in the first three categories the reliability as well as the share of real matchers found exceeded 60%. It is not as good result as in case of [10], where for the best identified similarity measures the values often exceeded 75%, but none of the evaluated similarity measures focussed on structural similarity as precisely as in our case and, of course, the match tasks were quite different.

Considering the Overall parameter the worst results are again in case of `rep` and `res` categories, whereas in case of `SimAvg` and `rep` category the value is even negative. This denotes that the number of false positives exceeds the number of true positives and such measure is almost useless, since the post-match effort is too high.

In general the experiments show that with the proper tuning of weights based on reliable information on representative sample data, the corresponding similarity measure has much better characteristics than the commonly used average-producing ones. In addition, the idea can be used for any type similarity measure (i.e. measure focussing not only on structural similarity) and any type of relevant tuning data.

5.2 Behavior of GAS

Considering the above described tuning we can assume that the similarity search algorithm is precise enough. But we cannot estimate its behavior on a contracted graph, e.g. the number of iterations of GAS or the structure of the resulting database schema. Thus in the following experiments we focus especially on the adaptive strategy. Since the approach is proposed to enable a user to assign a particular mapping to a chosen schema fragment which is suitable for the actual application, though it can be highly inefficient in the general case, we do not analyze the efficiency of the resulting relational schema. It depends highly on the user requirements and thus such experiments would be rather useless.

For testing the behavior of the proposed algorithm we have again used the 98 real-world XML schemes divided into the 5 categories (see Table 1). In experiments we use a slight modification of GAS (Algorithm 4) which enables to compare its behavior within the categories, where the annotated fragments are represented using a separate testing set of schema fragments consisting of 5 data-centric, 5 document-centric, 3 relational, and 3 DNA real-world schema fragments. Table 4 shows results of characteristics of the algorithm applied on all the sample fragments per each category.

Characteristic	**dat**	**doc**	**ex**	**rep**	**res**
Average number of iterations	2.7	3.9	2.9	4.1	4.3
Average % of not annotated nodes	2.1	53.4	13.5	25.6	31.1
% of fully contracted schemes	93.7	22.2	81.1	0.0	28.6

Table 4. General characteristics per category

As we can see, the algorithm has quite reasonable behavior. Firstly, the number of iterations is not an extreme one – the algorithm is able to perform more than one contraction (i.e. not only the BAS algorithm is applied) and, on the other hand, there are no extreme values with regard to usual depth or number of elements in the schemes (see Table 1). From the other two characteristics it is obvious that the schemes are not usually fully contracted (although it depends highly on the particular category), i.e. the storage strategies are not determined for the whole schema. This indicates that the default mapping strategy s_{def} should be still specified. If we compare the average number of iterations with the percentage of fully contracted schemes, it is surprising that schemes with the lower amount of contractions are fully contracted more often. It is probably caused by the fact, that the two categories, i.e. `dat` and `ex`, usually contain schemes with much simpler structure than, e.g., the `doc` one, or much regular than, e.g., the `res` one.

Next set of performed tests analyzed the behavior of the algorithm in particular iterations, especially the percentage of annotated nodes at each iteration, as depicted by graphs in Figure 14. As we can observe, the percentage of annotated nodes is usually highest in the first iteration, i.e. using the BAS algorithm, and then, with the decreasing number of nodes, rapidly decreases too. The only exceptions are the `rep` category, where the percentage grows up to third iteration and the `res` category, where it later slowly grows up to seventh iteration. It is probably caused by less regular structure than in the other three cases as well as previously mentioned lower number of sample XML schemes and thus less precise of tuning of the similarity measure.

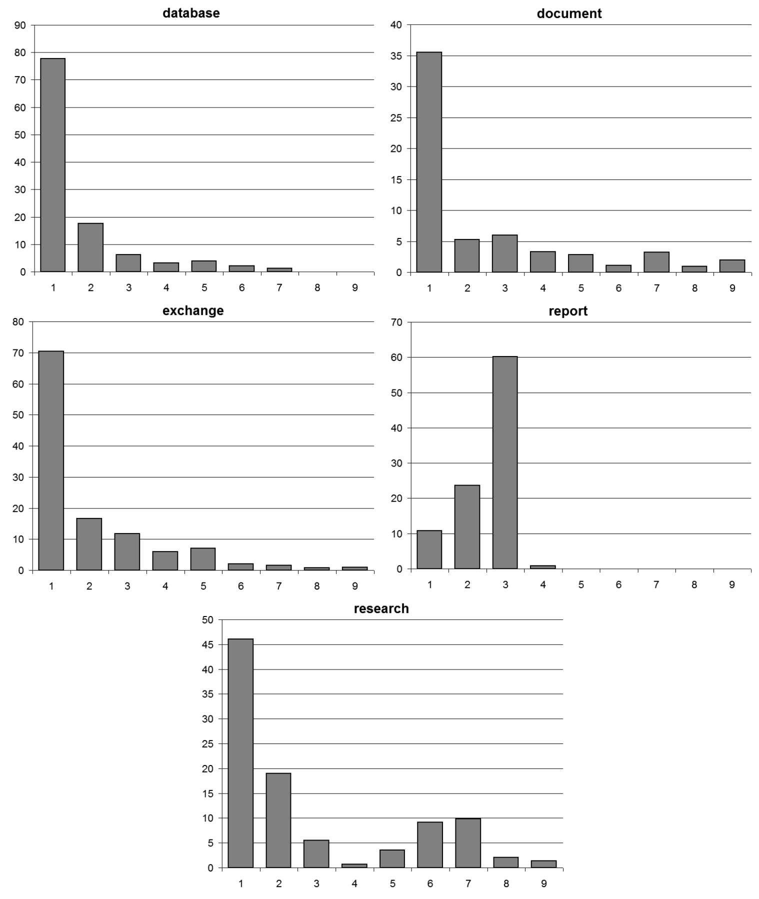

Fig. 14. Average percentage of annotated nodes at each iteration

The last set of performed tests analyzed the relation between the types of schema fragments and the iterations. The finding is that various types of schema fragments appear "randomly", regardless the iteration. This indicates that the algorithm does not provide degenerated schemes, such as, e.g., a schema where all the annotations correspond to a single schema fragment.

The results of the experiments show that the proposed approach is able to exploit the user-given information more deeply and find more appropriate mapping strategy for not annotated schema parts than s_{def}. When applied on real-world XML schemes and schema fragments, the algorithm behaves quite reasonably, though it is not usually able to annotate the given schema fully. This indicates that the default mapping strategy s_{def} is still important.

6 Conclusion

The main aim of this paper was to illustrate that since the idea of database-based XML processing methods is still up-to-date, the techniques should and can be further enhanced. On this account we have proposed a user-driven mapping algorithm which is able to exploit the user given information, i.e. schema annotations, more deeply and, at the same time, to find the mapping strategy for the not annotated parts more efficiently – using an adaptive approach. Using an experimental implementation we have shown that the approach has promising results. As a "side effect" can be considered a proposal and experimental evaluation of a similarity measure and its tuning.

A possible further improvement can be an exploitation of the semantic of element and/or attribute names. The similarity can be searched not only on structural level, but using a kind of thesaurus or appropriate user-given information. Although our proposal focuses mainly on structural similarities related to efficiency of database processing, it is at least worth testing whether the semantic of the names carries additional important information useful for this purpose too. Next interesting task could be also a combination of our approach with a cost-driven one, i.e. an exploitation of both user-given annotations and a sample set of XML data and XML queries together. As we have already mentioned, the key disadvantage is in the amount of required input data. But, on the other hand, the combination of the two approaches could bring interesting results and ensure full annotation of the schema. And, last but not least, the key enhancing lies in dynamic adaptability of the system [19]. This challenging but non-trivial task would solve the remaining disadvantage of the adaptive methods – the fact that the schema is adapted only once, at the beginning, but not in case the application changes.

Acknowledgement

This work was supported in part by the National Programme of Research (Information Society Project number 1ET100300419).

References

1. *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition).* W3C, August 2002. `http://www.w3.org/TR/xhtml1/`.

2. S. Amer-Yahia. *Storage Techniques and Mapping Schemas for XML*. Technical Report TD-5P4L7B, AT&T Labs-Research, 2003.
3. S. Amer-Yahia and M. Fernandez. *Overview of Existing XML Storage Techniques*. AT&T Labs-Research, 2001.
4. A. Balmin and Y. Papakonstantinou. Storing and Querying XML Data Using Denormalized Relational Databases. *The VLDB Journal*, 14(1):30–49, 2005.
5. R. Bartak. *On-Line Guide to Constraint Programming*. 1998. `http://kti.mff.cuni.cz/~bartak/constraints/`.
6. E. Bertino, G. Guerrini, and M. Mesiti. A Matching Algorithm for Measuring the Structural Similarity between an XML Document and a DTD and its Applications. *Inf. Syst.*, 29(1):23–46, 2004.
7. P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition)*. W3C, October 2004. `www.w3.org/TR/xmlschema-2/`.
8. P. Bohannon, J. Freire, P. Roy, and J. Simeon. From XML Schema to Relations: A Cost-based Approach to XML Storage. In *ICDE '02: Proc. of the 18th Int. Conf. on Data Engineering*, pages 64–75, Washington, DC, USA, 2002. IEEE Computer Society.
9. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C, September 2006. `http://www.w3.org/TR/REC-xml/`.
10. H. H. Do and E. Rahm. COMA – A System for Flexible Combination of Schema Matching Approaches. In *VLDB '02: Proc. of the 28th Int. Conf. on Very Large Data Bases*, pages 610–621, Hong Kong, China, 2002. Morgan Kaufmann Publishers Inc.
11. F. Du, S. Amer-Yahia, and J. Freire. ShreX: Managing XML Documents in Relational Databases. In *VLDB '04: Proc. of the 30th Int. Conf. on Very Large Data Bases*, pages 1297–1300, Toronto, ON, Canada, 2004. Morgan Kaufmann Publishers Inc.
12. D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDMBS. *IEEE Data Eng. Bull.*, 22(3):27–34, 1999.
13. J. H. Holland. *Adaptation in Natural and Artifical Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.
14. ISO/IEC 9075-14:2003. *Part 14: XML-Related Specifications (SQL/XML)*. International Organization for Standardization, 2006.
15. S. Kirkpatrick, C. D. Gerlatt Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.
16. M. Klettke and H. Meyer. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. In *Selected papers from the 3rd Int. Workshop WebDB '00 on The World Wide Web and Databases*, pages 151–170, London, UK, 2001. Springer-Verlag.
17. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *VLDB '01: Proc. of the 27th Int. Conf. on Very Large Data Bases*, pages 49–58, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
18. I. Mlynkova and J. Pokorny. From XML Schema to Object-Relational Database – an XML Schema-Driven Mapping Algorithm. In *ICWI '04: Proc. of the 3rd IADIS Int. Conf. WWW/Internet*, pages 115–122, Madrid, Spain, 2004. International Association for Development of the Information Society.
19. I. Mlynkova and J. Pokorny. Adaptability of Methods for Processing XML Data using Relational Databases – the State of the Art and Open Problems. *International Journal of Computer Science and Applications*, 4(2):43–62, 2007.

20. I. Mlynkova and J. Pokorny. *Exploitation of Similarity and Pattern Matching in XML Technologies.* Technical report 2006/13. Charles University, Prague, Czech Republic, November 2006. `http://kocour.ms.mff.cuni.cz/~mlynkova/doc/tr2006-13.pdf`.
21. I. Mlynkova, K. Toman, and J. Pokorny. Statistical Analysis of Real XML Data Collections. In *COMAD '06: Proc. of the 13th Int. Conf. on Management of Data*, pages 20–31, New Delhi, India, 2006. Tata McGraw-Hill Publishing Company Limited.
22. P. K.L. Ng and V. T.Y. Ng. Structural Similarity between XML Documents and DTDs. In *ICCS '03: Proc. of the Int. Conf. on Computational Science*, pages 412–421, Berlin, Heidelberg, 2003. Springer.
23. A. Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *WebDB '02: Proc. of the 5th Int. Workshop on the Web and Databases*, pages 61–66, Madison, Wisconsin, USA, 2002. ACM Press.
24. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, 2001.
25. 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 *VLDB '99: Proc. of the 25th Int. Conf. on Very Large Data Bases*, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
26. M. Smiljanic, M. van Keulen, and W. Jonker. Using Element Clustering to Increase the Efficiency of XML Schema Matching. In *ICDEW '06: Proc. of the 22nd Int. Conf. on Data Engineering Workshops*, pages 45–54, Los Alamitos, CA, USA, 2006. IEEE Computer Society Press.
27. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures (Second Edition).* W3C, October 2004. `www.w3.org/TR/xmlschema-1/`.
28. W. Xiao-ling, L. Jin-feng, and D. Yi-sheng. An Adaptable and Adjustable Mapping from XML Data to Tables in RDB. In *Proc. of the VLDB '02 Workshop EEXTT and CAiSE '02 Workshop DTWeb*, pages 117–130, London, UK, 2003. Springer-Verlag.
29. Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity Metric for XML Documents. In *FGWM '03: Proc. of Workshop on Knowledge and Experience Management*, Karlsruhe, Germany, 2003.
30. S. Zheng, J. Wen, and H. Lu. Cost-Driven Storage Schema Selection for XML. In *DASFAA '03: Proc. of the 8th Int. Conf. on Database Systems for Advanced Applications*, pages 337–344, Kyoto, Japan, 2003. IEEE Computer Society.