Schema Profiling of Document-Oriented Databases[☆],[☆☆]

Enrico Gallinucci, Matteo Golfarelli, Stefano Rizzi[*]

DISI – University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
CINI, Via Salaria 113, 00198 Roma, Italy

Abstract

In document-oriented databases, schema is a soft concept and the documents in a collection can be stored using different local schemata. This gives designers and implementers augmented flexibility; however, it requires an extra effort to understand the rules that drove the use of alternative schemata when sets of documents with different —and possibly conflicting— schemata are to be analyzed or integrated. In this paper we propose a technique, called schema profiling, to explain the schema variants within a collection in document-oriented databases by capturing the hidden rules explaining the use of these variants. We express these rules in the form of a decision tree (schema profile). Consistently with the requirements we elicited from real users, we aim at creating explicative, precise, and concise schema profiles. The algorithm we adopt to this end is inspired by the well-known C4.5 classification algorithm and builds on two original features: the coupling of value-based and schema-based conditions within schema profiles, and the introduction of a novel measure of entropy to assess the quality of a schema profile. A set of experimental tests made on both synthetic and real datasets demonstrates the effectiveness and efficiency of our approach.

Keywords: NoSQL, Document-Oriented Databases, Schema Discovery, Decision Trees

1. Introduction

Recent years have witnessed the progressive erosion of the relational DBMS predominance to the benefit of DBMSs based on different representation models (e.g., document-oriented and graph-based). Most new models adopt a *schemaless* representation for data, which does not mean that data are stored *without* a

[☆] This work was partly supported by the EU-funded project TOREADOR (contract n. H2020-688797).

[*] Corresponding author

Email addresses: `enrico.gallinucci2@unibo.it` (Enrico Gallinucci), `matteo.golfarelli@unibo.it` (Matteo Golfarelli), `stefano.rizzi@unibo.it` (Stefano Rizzi)

schema, but rather that schema is a soft concept and that the instances referring to the same concept in the same collection can be stored using different "local" schemata to better fit the specific features of each instance. So, while relational databases are still widely used in companies to store the stably-structured portions of corporate data, schemaless databases are preferred for storing heterogeneous data with variable schemata and structural forms, such as those located in so-called *data lakes*.

Typical schema variants that can be found within a collection consist in missing or additional attributes, in different names or types for an attribute, and in different structures for instances (obtained for example by moving a set of attributes into a sub-structure). These variants can either be created to uniformly store and handle instances with specific features, or be a consequence of the evolutions of the data management applications made in time to handle emerging requirements or to recover from errors occurred during schema design.

Unfortunately, the absence of a unique, well-defined schema turns to a disadvantage when moving from operational applications to analytical applications and business intelligence. Business intelligence analyses typically involve large sets of data, so a single analysis often involves instances with different —and possibly conflicting— schemata. In turn, this requires some extra effort to understand the rules that drove the use of alternative schemata, and an integration activity to identify a common schema to be adopted for analysis. These tasks are even harder when no detailed documentation is available, because the analyst has to search for the necessary knowledge either in the code that manages data or in the data themselves.

1.1. Approach Overview

In this paper we propose a technique, called BSP (*Build Schema Profile*), to explain the schema variants within a collection in document-oriented databases by capturing the hidden rules explaining the use of these variants. We call this activity *schema profiling*. Schema profiling is beneficial in different contexts:

- when trying to decode the behavior of an undocumented application that manages a document-base;
- when carrying out a data quality project on schemaless data;
- when adopting a schema-on-read approach to query a document-oriented database [1, 2];
- when designing a data warehouse on top of a schemaless data source, for instance a corporate data lake.

Identifying the rules of schema usage is much like building a descriptive model in a classification problem. A *classifier* is a model that predicts to which of a set of classes a new observation belongs, based on a training dataset containing observations whose class membership is known. Besides for predicting, classifiers are also used to *describe* the rules for assigning a class to an observation based on the other observation features —which corresponds to our goal if we consider the schema of a document as its class. So we can rely on the existing literature on classification to build schema profiles; in particular, based on the requirements we collected from potential users of our approach (see Section 3), among the different types of classifiers we consider decision trees since: (i) they natively provide an easy-to-understand representation of the schema usage rules; (ii) the tree-based representation is the one spontaneously adopted by the interviewed users; and (iii) the training approach for decision trees can be modified to accommodate the changes necessary to the context of this paper.

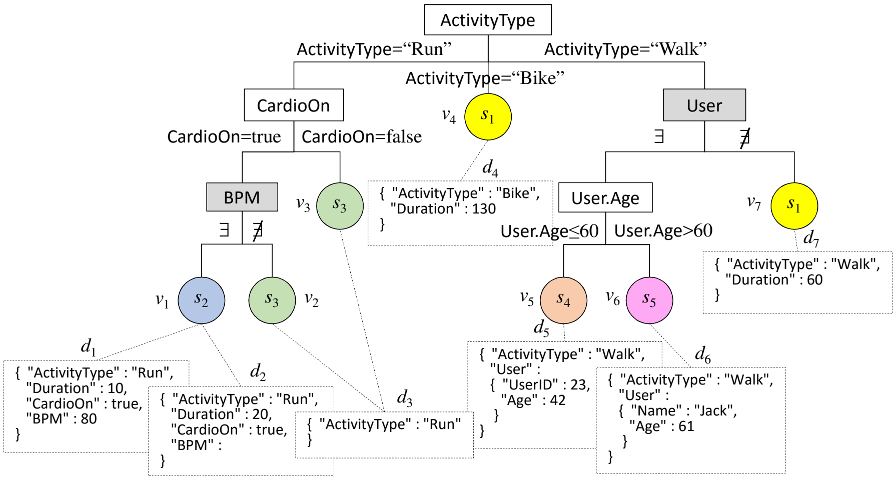
Straightly reusing traditional decision trees for schema profiling would mean classifying documents based on the *values* of their attributes only. However, this would often lead to trees where a single rule serves different classes (i.e., different schemata are explained by the same rule), which would give an imprecise information. To address this issues, in BSP documents are also classified using *schema-based conditions* related to the presence or absence of attributes. To better understand this point, consider for example Figure 1, showing a portion of a decision tree (which we call *schema profile*) built in the domain of physical fitness to profile a collection of documents generated by training machines or by their users through mobile applications. Each internal node in the tree is associated with a document attribute a and can express either a value-based condition (white box; each outgoing edge is related to one or more values of a, e.g., $\mathsf{User.Age} < 60$) or a schema-based condition (grey box; the two outgoing edges represent the presence or absence of a in the document, e.g., $\exists\mathsf{BPM}$). Each path in the tree models a *rule*; it leads to a leaf (represented as a circle) that corresponds to a schema found for the documents that meet all the conditions expressed along that path (document examples are shown in dashed boxes). So, for instance, schema s_2 is used in all the documents for which $\mathsf{ActivityType}$ = "Run", $\mathsf{CardioOn}$ = true, and field BPM is present (independently of its value, or even if a value is missing).

Another drawback of traditional decision trees is that they often give several rules for the same class. While this may be correct for some specific collections (e.g., schema s_1 in Figure 1 appears in two leaves, i.e., it is explained by two different rules), in general we wish to keep the number of rules to a minimum aimed at giving users a more concise picture of schema usage. This is achieved in BSP by adopting a novel measure of entropy to be coupled with the one typically used to characterize and build decision trees.

1.2. Contributions and Outline

The original contributions of this paper are:

- An analysis of the desiderata of real users that inspired BSP (Section 3).
- A novel measure of entropy, called *schema entropy*, which contributes to assess the quality of a schema profile (Section 5).
- An algorithm that implements a divisive approach to deliver precise, concise, and explicative schema profiles by mixing value-based and schema-



d_1
```
{ "ActivityType" : "Run",
  "Duration" : 10,
  "CardioOn" : true,
  "BPM" : 80
}
```

d_2
```
{ "ActivityType" : "Run",
  "Duration" : 20,
  "CardioOn" : true,
  "BPM" :
}
```

d_3
```
{ "ActivityType" : "Run"
}
```

d_4
```
{ "ActivityType" : "Bike",
  "Duration" : 130
}
```

d_5
```
{ "ActivityType" : "Walk",
  "User" :
  { "UserID" : 23,
    "Age" : 42
  }
}
```

d_6
```
{ "ActivityType" : "Walk",
  "User" :
  { "Name" : "Jack",
    "Age" : 61
  }
}
```

d_7
```
{ "ActivityType" : "Walk",
  "Duration" : 60
}
```

Figure 1: A schema profile in the physical fitness domain

based conditions to better capture the rules explaining the use of different schemata within a collection (Section 6).

- A set of experimental results on synthetic and real datasets (Section 7).

The paper is completed by Section 2, which discusses related work, Section 4, which provides the necessary formal background, and Section 8, which draws the conclusions.

2. Related Work

In this section we discuss the research areas mainly related to our work. In particular, in Subsection 2.1 we summarize the approaches for schema discovery on semi-structured data collections, while in Subsection 2.2 we discuss the main techniques for schema matching.

2.1. Schema Discovery

The task of schema discovery is not new to the research world. Early work focused on finding ways to describe semi-structured data retrieved from web pages and were mainly based on the Object Exchange Model (OEM). The general idea was to automatically derive a concise [3] or approximate [4] labeled graph model to allow efficient querying of data. In the latter group, approximation was achieved by applying clustering algorithms on the data, so that a different model could be defined for each syntactically-similar group. As XML became a standard for data exchange on the web, researchers had to deal with the widespread lack of DTDs and XSDs for XML documents. Work such as

[5, 6, 7] focused on extracting the regular expressions that described the contents of the elements in a set of XML documents, aimed at helping software tools in processing XML documents or in integrating data through schema matching.

With the replacement of XML with JSON, similar issues are now being addressed on this new standard, with the goal of capturing and representing the intrinsic variety of schemata within a collection of JSON objects. Izquierdo and Cabot [8] propose to build a unique schema for the JSON objects returned by a single API service; their goal is then to create a unified model to represent the matches between the schemata derived from different API services. Klettke et al. [9] propose two different graph structures that model the union of all the attributes in a collection of JSON objects, aimed at measuring the heterogeneity of a collection and at detecting attributes with low support for data quality investigations. Ruiz et al. [10] propose a reverse engineering process to derive a versioned schema model for a collection of JSON objects: instead of considering the mere union of the attributes within a nested object, every intensional variation is stored as a different version of such object. Wang et al. [11] adopt a clustering technique to identify the smallest set of core attributes (called *skeleton*) by grouping similar schemata corresponding to specific business objects. Whereas the skeleton facilitates the recognition of the underlying schemata by the user, the assumption that schema-similarity is the metric that characterizes the different object types is potentially misleading; indeed, this is not always true in our experience.

The main features of the approaches mentioned above are summarized in Table 1 and compared to those of BSP. Though BSP shares with the others some techniques, its goal is completely different. The ultimate goal of the previous work is to provide either a concise [9, 11] or a comprehensive [8, 10] way to *describe* the intensional aspects of the documents, not only for an easier recognition of the schemata, but also to enable automated activities such as querying, integration, and validation. Conversely, the goal of BSP is to *explain* the schema variants. Besides, no other approach considers the extensional point of view to describe the different usages of schemata. Finally, while all previous approaches produce in output some form of (global, skeleton, or reduced) schema, BSP creates a schema profile that classifies schemata.

With the increasing diffusion of document-based stores, several tools are currently known to perform schema detection on the available DBMSs. Starting from MongoDB, a handful of free tools to help users in the analysis of the hidden schema have been designed by third-party developers, such as variety.js[1], schema.js[2], and mongodb-schema[3]. In ElasticSearch, the embedded functionality of *dynamic mapping*[4] automatically infers the schema to enable

[1] `http://github.com/variety/variety`

[2] `http://www.npmjs.com/package/schema-js`

[3] `http://github.com/mongodb-js/mongodb-schema`

[4] `http://www.elastic.co/guide/en/elasticsearch/reference/current/dynamic-mapping.html`

Table 1: A comparison of the main approaches to schema discovery with BSP

Approach	*Data format*	*Arrays*	*Attribute match*	*Output*	*Goal*
[5, 6, 7]	XML	yes	by name	regular expr.	describe global schema
[8]	JSON	no	by name+values	multi-schema	describe inter-schema matches
[9]	JSON	yes	by name	reduced graph	describe global schema
[10]	JSON	yes	none	multi-schema	describe schema variants
[11]	JSON	no	similarity	skeleton schema	describe main schema features
BSP	JSON	no	by name+type	schema profile	explain schema variants

search capabilities on the documents. In Couchbase, the Spark connector[5] can perform automatic schema inference that will enable Spark SQL operations. Also Apache Drill[6] dynamically infers the schema at query time —although it is used only internally and cannot be exported by the user. All these tools derive a unique schema which collects the union of the attributes found in the documents, possibly enriched by additional information such as the types and support of each attribute. Finally, it is worth mentioning that the JSON-schema initiative[7] has risen with the idea to provide standard specifications to describe JSON schemata; however, its adoption is still quite limited and restricted to validation software.

2.2. Schema Matching

In parallel to schema discovery, plenty of research have focused on the issues related to schema matching. Bernstein et al. [12] provide a comprehensive summary of the different techniques envisioned for generic schema matching, which ranges from the relational world to ontologies and XML documents. Other papers have also investigated the applications of mining techniques specifically to XML documents: for instance, Nayak and Iryadi [13] and Lee et al. [14] provide clustering algorithms to be applied to XML schemata, while Guerrini et al. [15] provides an overview of the different similarity measures that can be used to cluster XML documents.

An interesting branch of schema matching is the one that exploits contextual information in the data. For example, Bohannon et al. [16] propose an enhancement of schema matching algorithms by extending matches (which refer to table attributes) with selection conditions, which enables to identify the cases in which the match is actually valid. The principle of analyzing instance values is used also in [17] and [18], which apply machine learning techniques to

[5] `http://github.com/couchbase/couchbase-spark-connector/wiki/Spark-SQL`
[6] `http://drill.apache.org`
[7] `json-schema.org`

schema matching. Interestingly, while these approaches use contextual information to identify schema matching conditions, we use contextual information for the opposite reason, i.e., to justify schema heterogeneity.

3. Requirements for Schema Profiling

The first stage of our work has been devoted to elicit user requirements for schema profiling with reference to the application domains of two projects in which we were involved: the one of a company selling fitness equipment, whose documents (corresponding to two datasets named RD1 and RD2 in Section 7) contain the registrations of workout sessions, and the one of a software development company, whose documents (dataset RD3) contain the log of the errors generated by the deployed applications. In both domains, elicitation was conducted as follows:

1. **Interview**. During these half-day sessions, we conducted semi-structured interviews with the users to understand their goals, the type of information they require, and the visualization format they prefer for the results.

    - In the fitness domain we interviewed three users: the database administrator, the application manager, and a data scientist. Both the schemata and the data management applications were internally developed, so the users were supposed to have full control and good knowledge of them.
    - In the software development domain we interviewed two users: the chief software developer and a junior developer. They use a third-party repository for error logging, so in this case the users had little control on the schema variants and on the data management applications.

    At the end of the interview, we asked both user groups to describe the schema variants they were aware of, and the reasons that led to using each specific schema. No suggestions were given by us as to how these descriptions should be expressed.

2. **Description**. After one week we had a second half-day session, during which each user group delivered its descriptions as required and discussed it with us. In both cases a tree was delivered where each node represented a condition, much like in Figure 1. Most conditions were given on instances (e.g., $\mathsf{Date} \leq 2016$), while some were given on schemata (i.e., presence/absence of some attributes in a document). Schema-based conditions were said to be used less because, though they obviously *describe* the difference between schemata, they do not provide any *explanation* of the the reason of this difference.

3. **Check**. At this stage we checked the user descriptions against the documents. For both domains the descriptions were incomplete (i.e., users

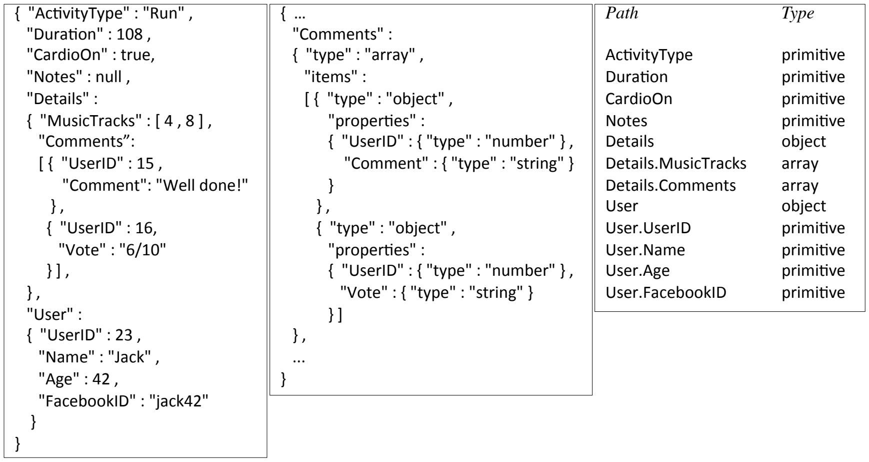
were not able to specify all the rules that precisely characterize each single schema), and in some cases even wrong (i.e., the rules provided did not correctly separate the documents according to their schemata).

The main hints we obtained are summarized below:

- Users need an easy-to-read, graphical, and compact schematization of schema usage.
- Value-based conditions are preferred to schema-based ones. From the users' point of view, while a schema-based condition merely acknowledges the difference between two schemata, a value-based condition explains this difference in terms of the values taken by an attribute. For instance, with reference to Figure 1, the schema-based condition on User states that *"The documents with schema s_1 differ from those with schema s_4 or s_5 in that they give no description of the user"*; conversely, the value-based condition on User.Age states that *"The documents with schema s_4 and those with schema s_5 differ because they are related to young and elderly users, respectively"*.
- Users tend to describe the conditions that, according to their domain knowledge, are most significant first. The importance of a condition is subjective to some extent, but it is typically related to the usage of different applications/functionalities rather than to the actual inter-schema similarity. Indeed, we found documents with very similar schemata that are actually quite unrelated (e.g., they log the errors of different software applications). This suggests to discard solutions, like [11], that reduce the total number of schemata by clustering similar schemata together, because the skeleton schemata they produce might lack some relevant attributes.
- Even for very experienced users, it is quite difficult to provide a precise and complete description of the conditions ruling the schema usage.

Based on these considerations, we claim that an automated approach to schema profiling is potentially beneficial and should have the following features:

- the result must take the form of a decision tree (called *schema profile* from now on) to provide a comprehensible description;
- the schema profile should be *explicative*, i.e., it should give priority to value-based conditions;
- the schema profile should be *precise*, i.e., it should accurately characterize each single schema;
- the schema profile should be *concise*, i.e., it should provide a small set of rules.



```
{ "ActivityType" : "Run" ,
  "Duration" : 108 ,
  "CardioOn" : true,
  "Notes" : null ,
  "Details" :
  { "MusicTracks" : [ 4 , 8 ] ,
    "Comments”:
    [ { "UserID" : 15 ,
        "Comment": "Well done!"
      } ,
      { "UserID" : 16,
        "Vote" : "6/10"
      } ] ,
  } ,
  "User" :
  { "UserID" : 23 ,
    "Name" : "Jack" ,
    "Age" : 42 ,
    "FacebookID" : "jack42"
  }
}
```

(a)

```
{ …
  "Comments" :
  { "type" : "array" ,
    "items" :
    [ { "type" : "object" ,
        "properties" :
        { "UserID" : { "type" : "number" } ,
          "Comment" : { "type" : "string" }
        }
      } ,
      { "type" : "object" ,
        "properties" :
        { "UserID" : { "type" : "number" } ,
          "Vote" : { "type" : "string" }
        } ]
  } ,
  …
}
```

(b)

Path	*Type*
ActivityType	primitive
Duration	primitive
CardioOn	primitive
Notes	primitive
Details	object
Details.MusicTracks	array
Details.Comments	array
User	object
User.UserID	primitive
User.Name	primitive
User.Age	primitive
User.FacebookID	primitive

(c)

Figure 2: A JSON document representing a training session (a), a portion of its JSON schema (b), and its r-schema (c)

4. Formal Background

The central concept of a document-oriented database is the notion of *document*, which encapsulates and encodes data in some standard format. The most widely adopted format is currently JSON, which we will use as a reference in this work.

Definition 1 (Document and Collection). *A* document *d is a JSON object. An* object *is formed by a set of* name/value *pairs, commonly referred to as* elements*. A* value *can be either a primitive value (i.e, a number, a string, or a Boolean), an array of values, an object, or* null*. Names cannot be repeated within the same object, but they can be repeated at different nesting levels. A collection D is a set of documents.*

An important feature of JSON is that arrays have no constraint on the type of their values, i.e., an array can simultaneously contain numbers, strings, other arrays, as well as objects with different internal structures. Figure 2.a shows a document representing a training session containing an array of objects with different schemata (element Comments).

The JSON schema initiative provides the specifications to define the schema of a document; however, as highlighted in the example in Figure 2, the resulting schemata are quite verbose and they provide a complex representation of arrays. Indeed, the schema of an array is defined as the ordered list of the schemata of its values; in other words, two arrays share the same schema only if they contain the same number of values and these values share the same schemata in the same order. This schema-matching criterion would basically lead to having a different schema for each single document, which would add unnecessary burden to our approach —also considering that the type variability of array elements is less relevant than that of the other attributes. Thus we adopt a more concise representation for the schema of a document, called *reduced schema*, which does not enter into the content of arrays, but simply denotes the presence of an array structure.

Definition 2 (R-Schema of a Document). *Given document d, the reduced schema (briefly,* r-schema*) of d, denoted $rs(d)$, is a set of* attributes*, each corresponding to one element in d. Attribute $a \in rs(d)$ is identified by a* pathname*, $path(a)$, and by a type, $type(a) \in \{\mathsf{primitive}, \mathsf{object}, \mathsf{array}\}$. While $path(a)$ is a string in dot notation reproducing the path of the element corresponding to a in d, $type(a)$ is the type of that element (type* primitive *generalizes numbers, strings, Booleans, and nulls).*

Note that, although r-schemata are defined as sets of attributes, their pathnames code the document structure (short of the internal structure of arrays).

Example 1. *Figure 2 shows a sample document, its JSON schema (as per the specifications of the JSON schema initiative), and its r-schema. Note how, in the r-schema, the complexity and heterogeneity of array* Comments *is hidden in attribute* Details.Comments *with generic type* array*.*

To put together the relevant information coded by different r-schemata, we define a sort of global schema for the documents within a collection:

Definition 3 (R-Schema of a Collection). *Let two attributes in the r-schemata of two documents be equal if they have the same pathname and the same type. Given collection D, we denote $S(D)$ the set of distinct r-schemata of the documents in D (). The* r-schema *of D is defined as*

$$rs(D) = \bigcup_{s \in S(D)} s = \bigcup_{d \in D} rs(d)$$

Given $s \in S(D)$, we denote with $|D|_s$ the number of documents in D with r-schema s.

Intuitively, $rs(D)$ includes all the distinct attributes that appear in the r-schemata of the documents in D.

The last concept we need to introduce are *schema profiles*.

Definition 4 (Schema Profile). *Let D be a collection. A* schema profile *for D is a directed tree T where each internal node (including the root) corresponds to some attribute $a \in rs(D)$ and can be either* value-based *or* schema-based*:*

- *a schema-based node has exactly two outgoing edges, labelled as $\exists$ and $\nexists$ respectively;*
- *a value-based node has two or more outgoing edges, each labelled with a condition expressed over the domain of a.*

Value-based nodes can only correspond to attributes of type primitive*. Given node v, we denote with $D_v \subseteq D$ the set of documents that meet all the conditions expressed by the nodes in the path from the root to v. In particular, a document d such that $a \notin rs(d)$ (missing attribute) or $d.a =$* null *(missing value) always meets all the conditions expressed by a value-based node corresponding to a.*

Intuitively, each internal node in a schema profile models a condition (on a value-based column of the dataset if the node is value-based, on a schema-based column if it is schema-based), and each path models a *rule* that includes a set of conditions for selecting one or more r-schemata. We also remark that Definition 4 can accommodate both binary and n-ary trees; in Section 5 we will show that binary trees are preferable to n-ary trees in our context.

The approach we adopt to deal with missing attributes and missing values in Definition 4 is consistent with the one used by the decision tree algorithm we chose, i.e., C4.5 [19] in its Weka implementation. In presence of a value-based node corresponding to a, such document d is considered to belong to two or more leaves; in other words, the sets D_{v_j} for $j = 1, \ldots, m$ are not necessarily disjoint. Algorithm C4.5 also adopts a weighting mechanism in these cases; specifically, when computing leaf cardinalities (namely, $|D_{v_j}|$ and $|D_{v_j}|_s$ in Section 5), a document d such that $d \in D_{v'} \cap D_{v''}$ is weighted depending on how the other documents are distributed in v' and v'' [19].

Example 2. *Figure 1 shows an n-ary schema profile with two schema-based nodes (*User *and* BPM*) and three value-based nodes (*ActivityType*,* CardioOn*, and* User.Age*). In this case, $D = \{d_1, \ldots, d_7\}$ and $S(D) = \{s_1, \ldots, s_5\}$. The schema profile has leaves $v_1, \ldots, v_7$, with $D_{v_1} = \{d_1, d_2\}$. Note that document d_3 belongs to both v_2 and v_3, since attribute* CardioOn *is missing.*

5. Quantifying Schema Profile Requirements

In Section 3 we claimed that, according to the users' feedback, a good-quality schema profile should be precise, concise, and explicative. In the following subsections we discuss how we quantitatively characterize the requirements of schema profiles; the criteria introduced here will be then used in the algorithm proposed in Section 6.

Algorithm C4.5 adopts a divisive approach to build the decision tree. Divisive approaches start from a single node that includes all the observations, and then iteratively split each node into two or more nodes that include subsets of observations. The split point is chosen in a greedy fashion by trying to maximize the quality of the classification. In our context, splits come in different flavors.

Definition 5 (Split). *Let D be a collection and $a \in rs(D)$ be an attribute in the r-schema of D. Let T be a schema profile for D and v be one of its leaves. A* split of v on a*, denoted $\sigma_a(v)$, transforms T into a schema profile T' where v is an internal node corresponding to a and has two or more new leaves as children. In particular, $\sigma_a(v)$ is a* schema-based split *if v is a schema-based node in T',* *and a* value-based split *if v is a value-based node in T'. Value-based splits are made as follows:*

- *If a is numeric, $\sigma_a(v)$ has two children and the corresponding edges are labelled with conditions $a \leq$ "val" and $a >$ "val", respectively.*
- *If a is categorical, $\sigma_a(v)$ can be either* binary *or* multi-way*. In the first case, it has two children and the corresponding edges are labelled with conditions $a =$ "val" and $a \neq$ "val", where "val" is a value of the domain of a. In the second case, it has one child for each value "val" in the domain of a and each corresponding edge is labelled with condition $a =$ "val".*

Based on this definition, in a schema-based split v has two children and the corresponding edges are labelled as $\exists$ and $\nexists$), while in a value-based split v has two or more children and the corresponding edges are labelled with a condition expressed over the domain of a. Note that, while any splitting value "val" can be adopted for value-based splits on numeric attributes and binary splits on categorical attributes, different values produce schema profiles with different qualities. The criteria we adopt for evaluating splits (namely, gain and loss) will be introduced in the following subsections, while in Section 6 we will propose an algorithm based on a function ($FindBestSplit$) that finds the best split for a given attribute by exhaustive search on its domain.

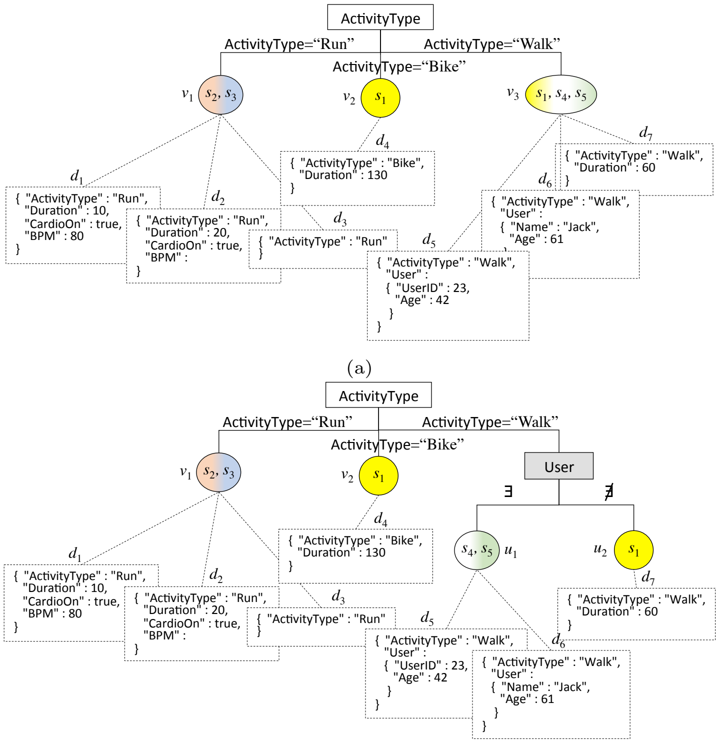
Example 3. *Figure 3 shows an example of schema-based split at an early stage of the building of the schema profile of Figure 1. Initially, T has three leaves v_1, v_2, v_3 (resulting from a multi-way, value-based split on the categorical attribute* ActivityType*). The schema-based split on attribute* User *creates a new schema profile where v_3 becomes a schema-based node with two children, u_1 and u_2. A further value-based split on the numeric attribute* User.Age *creates two additional leaves as depicted in Figure 1.*

5.1. Explicativeness

As mentioned in Section 3, we consider a schema profile to be explicative if it prefers value-based nodes over schema-based nodes, because the latter acknowledge that there is a difference between two r-schemata but do not really explain its reason. Indeed, a schema-based condition on an attribute a always partitions the documents based on the presence/absence of a so, in a sense, it merely explains itself. On the other hand, a value-based condition on the values of a does not partition the documents based on the presence/absence of a, and it always relates a to other attributes. For instance, in Figure 1, schemata s_4 and s_5 differ since they include attributes User.UserID and User.Name, respectively, and an explanation for this difference is given in terms of the values taken by attribute User.Age.

So, when comparing two schema profiles, we will evaluate explicativeness using the percentage of schema-based nodes; the lower this number, the more explicative the schema profile:

$$\%\ \textit{schema-based nodes} = \frac{\sharp\ \textit{schema-based nodes}}{\sharp\ \textit{nodes}}$$


ActivityType
ActivityType="Run"
ActivityType="Bike"
ActivityType="Walk"
v_1 s_2, s_3
v_2 s_1
v_3 s_1, s_4, s_5
d_1
{ "ActivityType" : "Run", "Duration" : 10, "CardioOn" : true, "BPM" : 80 }
d_2
{ "ActivityType" : "Run", "Duration" : 20, "CardioOn" : true, "BPM" : }
d_3
{ "ActivityType" : "Run" }
d_4
{ "ActivityType" : "Bike", "Duration" : 130 }
d_5
{ "ActivityType" : "Walk", "User" : { "UserID" : 23, "Age" : 42 } }
d_6
{ "ActivityType" : "Walk", "User" : { "Name" : "Jack", "Age" : 61 }
d_7
{ "ActivityType" : "Walk", "Duration" : 60 }
(a)
ActivityType
ActivityType="Run"
ActivityType="Bike"
ActivityType="Walk"
User
∃
∄
v_1 s_2, s_3
v_2 s_1
s_4, s_5 u_1
u_2 s_1
d_1
{ "ActivityType" : "Run", "Duration" : 10, "CardioOn" : true, "BPM" : 80 }
d_2
{ "ActivityType" : "Run", "Duration" : 20, "CardioOn" : true, "BPM" : }
d_3
{ "ActivityType" : "Run" }
d_4
{ "ActivityType" : "Bike", "Duration" : 130 }
d_5
{ "ActivityType" : "Walk", "User" : { "UserID" : 23, "Age" : 42 } }
d_6
{ "ActivityType" : "Walk", "User" : { "Name" : "Jack", "Age" : 61 } }
d_7
{ "ActivityType" : "Walk", "Duration" : 60 }

(b)

Figure 3: Schema profile before (a) and after (b) a schema-based split

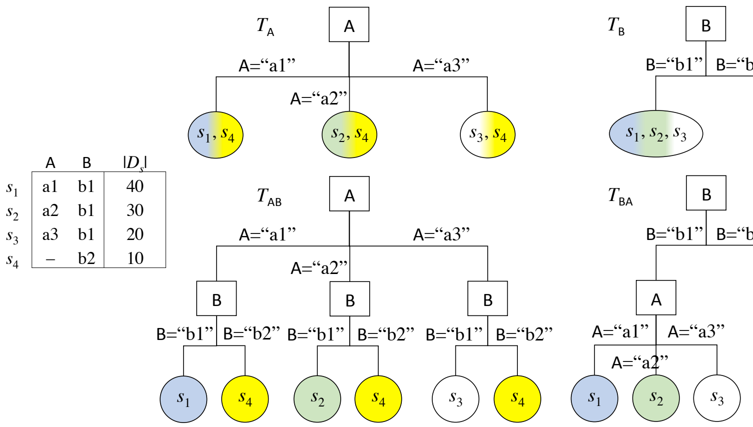
5.2. Precision

A distinguishing feature of divisive approaches is the function they adopt to quantify the "purity" of the leaves where observations are classified, where a leaf is said to be *pure* if all its observations share the same class. The most common function used to this end is *entropy* [20], whose definition —properly contextualized to our application domain— we include below.

Definition 6 (Entropy and Gain). *Let D be a collection, $S(D)$ be the set of distinct r-schemata of the documents in D, and T be a schema profile for D with leaves $v_1, \ldots, v_m$. The entropy of leaf v_j is*

$$entropy(v_j) = -\sum_{s \in S(D_v)} \frac{|D_{v_j}|_s}{|D_{v_j}|} \log \frac{|D_{v_j}|_s}{|D_{v_j}|} \tag{1}$$

where $\frac{|D_{v_j}|_s}{|D_{v_j}|}$ is the probability of r-schema s within leaf v_j. Leaf v_j is said to be pure *if $entropy(v_j) = 0$. The* entropy *of T is then defined as the weighted sum*



	A	B	$\lvert D_s \rvert$
s_1	a1	b1	40
s_2	a2	b1	30
s_3	a3	b1	20
s_4	–	b2	10

Figure 4: A collection (on the left) and four possible schema profiles (see Example 4)

of the entropies of the leaves of T:

$$entropy(T) = \sum_{j=1}^{m} \frac{|D_{v_j}|}{|D|} \cdot entropy(v_j) \tag{2}$$

where $\frac{|D_{v_j}|}{|D|}$ is the probability of leaf v_j. Let $\sigma_a(v_j)$ be a split that creates two or more leaves, $u_1, \ldots, u_r$, resulting into a schema profile T'; the gain *related to $\sigma_a(v_j)$ is defined as the difference between the entropy of T and the one of T':*

$$gain(\sigma_a(v_j)) = entropy(T) - entropy(T') =$$
$$= \frac{|D_{v_j}|}{|D|} \cdot entropy(v_j) - \sum_{k=1}^{r} \frac{|D_{u_k}|}{|D|} \cdot entropy(u_k) \tag{3}$$

Entropy is strictly related to precision as informally defined in Section 3: within a schema profile with null entropy, all the documents included in each leaf share the same r-schema, thus the schema profile has maximum precision.

Example 4. *Let D be a collection with 100 documents, 4 r-schemata $s_1, \ldots, s_4$, and two attributes, A and B. The values of the attributes for the different r-schemata are listed in Figure 4 (symbol "–" means "any value"), together with four possible schema profiles. A degenerate schema profile T_0 that includes a single node v_0 has entropy $entropy(T_0) = entropy(v_0) = 1.85$ (because $D_{v_0} \equiv D$). The multi-way splits of v on A and B have $gain(\sigma_{\mathsf{A}}(v_0)) = 1.39$ and $gain(\sigma_{\mathsf{B}}(v_0)) = 0.47$, and produce schema profiles T_{A} and T_{B}, respectively, with $entropy(T_{\mathsf{A}}) = 0.46$ and $entropy(T_{\mathsf{B}}) = 1.38$. Both schema profiles T_{AB} and T_{BA} have null entropy.*

Entropy tends to decrease with each split; the higher the gain (i.e., the decrease in entropy), the more convenient the split. It is well-known [21] that gain-based criteria are biased and tend to privilege multi-way splits on attributes with several values. A classical solution to this problem is to consider binary splits only, or to add a weighting factor to normalize the gain for different attributes (so-called *gain ratio* criterion) [22].

5.3. Conciseness

Entropy is focused on node purity, hence its minimization often leads to split observations of the same class among several leaves; this is more frequent when the number of classes is high [23], as normally happens in our context. While this is a secondary problem in generic classification problems, where the precision of the resulting model is more important than its readability, it becomes critical in schema profiling since it conflicts with the conciseness requirement. Indeed, in our context, each r-schema might end up for being explained by a wide set of rules, thus precluding users from getting a concise picture of schema usage. For instance, with reference to Example 4, T_{BA} is clearly the schema profile that best represents the collection, with each r-schema being reached by a single path in the tree. Nevertheless, the tree chosen by an entropy-based algorithm would be T_{AB}, where s_4 is doubled, because of the higher gain when splitting T_0 on A.

To evaluate conciseness of schema profiles, in this section we propose a measure called *schema entropy*. As stated in Section 3, our requirement is reduce the number of rules provided by maximizing the cohesion of the documents that share the same r-schema —a maximally concise schema profile is one where there is a single rule for each r-schema. So we invert the original definition of entropy to relate it to the purity of the r-schemata instead of the purity of the leaves: in terms of entropy, a leaf is pure if it contains only documents with the same r-schema; in terms of schema entropy, an r-schema is pure if all its documents are in the same leaf. The schema entropy of a degenerate schema profile where all the documents are included into a single node (the root) is 0; clearly, when a split is made, the schema entropy can never decrease, so the concept of gain is replaced by that of *loss* (the lower the loss, the more convenient the split).

Definition 7 (Schema Entropy and Loss). *Let D be a collection, $S(D)$ be the set of distinct r-schemata of the documents in D, and T be a schema profile for D with leaves $v_1, \ldots, v_m$. The schema entropy of r-schema $s \in S(D)$ is*

$$sEntropy(s) = \sum_{j=1}^{m} \frac{|D_{v_j}|_s}{|D|_s} \log \frac{|D_{v_j}|_s}{|D|_s} \tag{4}$$

The schema entropy *of T is then defined as*

$$sEntropy(T) = -\sum_{s \in S(D)} \frac{|D|_s}{|D|} \cdot sEntropy(s) \tag{5}$$

Let $\sigma_a(v_j)$ be a split resulting into schema profile T'; the loss *related to $\sigma_a(v_j)$ is defined as:*

$$loss(\sigma_a(v_j)) = sEntropy(T') - sEntropy(T) \tag{6}$$

Definition 7 implies that the loss of split $\sigma_a(v_j)$ is 0 iff, for each r-schema $s \in S(D_{v_j})$, all the documents with r-schema s belonging to D_{v_j} are put into a single leaf of T'. As a consequence, all schema-based splits have null loss.

Example 5. *With reference to Example 4, T_0 has null schema entropy. It is $loss(\sigma_{\mathsf{A}}(v_0)) = 0.16$ and $loss(\sigma_{\mathsf{B}}(v_0)) = 0$; since all the following splits have null loss, it is $sEntropy(T_{\mathsf{AB}}) = 0.16$ and $sEntropy(T_{\mathsf{BA}}) = 0$. Therefore, T_{BA} is the most convenient from the point of view of schema entropy.*

Let m be the number of leaves and n the number of r-schemata; we note that:

- The schema entropy of a schema profile T can be 0 only if $m \leq n$ (because clearly, if $m > n$, at least one r-schema appears in more than one leaf).
- The entropy of a schema profile T can be 0 only if $m \geq n$.

Although these observations apparently suggest that entropy and schema entropy are conflicting, actually their goals are not mutually exclusive. Indeed, splitting a node so that documents with the same r-schema are kept together, means putting documents with different r-schemata in separate children; so, the splits determining low or null loss tend to yield a high gain as well. In Section 6 we will show how BSP builds on both entropy and schema entropy to achieve a trade-off between conciseness and precision of schema profiles.

We close this section with an important remark. As already stated, our definition of schema profile (Definition 4) accommodates both binary and n-ary trees —depending on whether binary or multi-way splits are allowed. However, we observe that an n-ary schema profile can always be translated into binary form without changing its entropy and schema entropy (because they only depend on how documents are partitioned among the leaves). Besides, multi-way splits on high-cardinality categorical attributes produce a strong fragmentation of r-schemata into leaves and an undesired increase in schema entropy. For these reasons, in the following we will focus on binary schema profiles.

6. Building Schema Profiles

As already mentioned, our algorithm is inspired to C4.5 [19] in its Weka implementation (named J48). C4.5 implements a divisive approach that starts by creating a single node that includes all the observations in the dataset; as such, according to Definition 6, this root node has maximum entropy. Then the algorithm iteratively splits each leaf into two new leaves that include subsets of observations; since the final goal is to minimize the entropy, at each iteration the split that maximizes the gain is greedily chosen. The algorithm stops when either all the leaves are pure or all the splits are statistically irrelevant.

In our domain, recalling the user requirements illustrated in Section 3, we can informally state our final goal as follows:

Given collection D find, among the schema profiles for D with null entropy, one that (i) has minimum schema entropy on the one hand, and (ii) has the minimum number of schema-based nodes on the other.

Null entropy ensures that each leaf includes documents belonging to one r-schema only (i.e., the schema profile is *precise*), schema-entropy minimization calls for having a single rule to explain each r-schema (the schema profile is *concise*), while minimization of schema-based nodes ensures that most conditions are meaningful (the schema profile is *explicative*). Obviously, to fit this goal, the splitting and stop strategies of the C4.5 algorithm have to be modified by considering schema entropy on the one hand, and by taking into account the distinction between schema-based and value-based splits on the other.

A solution for our problem can be always found since, as no two r-schemata with exactly the same attributes exist, any schema profile can be extended using schema-based nodes until each leaf includes documents belonging to one r-schema only (i.e., the schema profile has null entropy). On the other hand, the (trivial) schema profiles including only schema-based nodes have null schema entropy but they do not meet subgoal (ii). Indeed, subgoals (i) and (ii) may be conflicting; to rule the trade-off, in BSP we allow schema-based splits only if no value-based split satisfies the following quality criterion:

Definition 8 (Valid Split). *A value-based (binary) split $\sigma_a(v)$ is* valid *if $loss(\sigma_a(v)) \leq \epsilon$ and $gain(\sigma_a(v)) \geq \omega$, where $\epsilon, \omega \geq 0$ are thresholds.*

Based on Definitions 6 and 7, it is easy to verify that $\epsilon \in [0..log|D|]$ and $\omega \in [0..log|S(D)|]$. There is not an ideal all-purpose setting for these threshold; as we will show in Section 7, through ϵ and ω users can fine-tune the trade-off between subgoals (i) and (ii) so as to obtain satisfactory schema profiles depending on the specific features of the collection. More specifically: the higher ϵ, the more the user is privileging schema profiles with a few schema-based nodes —i.e., explicative ones; the higher ω, the more the user is favoring schema profiles whose value-based nodes considerably decrease entropy —i.e., concise ones.

Consistently with the final goal stated above, we define a criterion for comparing two splits as follows:

Definition 9. *Given two splits $\sigma(v)$ and $\sigma'(v)$ (possibly on different attributes), we say that $\sigma(v)$ is* better *than $\sigma'(v)$ (denoted $\sigma(v) \prec \sigma'(v)$) if either (i) $loss(\sigma(v)) < loss(\sigma'(v))$, or (ii) $loss(\sigma(v)) = loss(\sigma'(v))$ and $gain(\sigma(v)) > gain(\sigma'(v))$.*

The pseudocode of Algorithm 1 implements BSP. It is a recursive procedure that splits a generic leaf v of the tree T representing the schema profile; it is initially called with a degenerate tree consisting of one single node, and stops when all leaves are pure. Value-based splits are tried first (lines 3 to 6); candidate attributes for splitting are those that are present in at least one of the documents in D_v. For each candidate attribute a, function $FindBestSplit$ finds the best binary split by exhaustively computing gain and loss for each possible

Algorithm 1 BuildSchemaProfile (BSP)

```
Require: $T$, a schema profile; $v$, the leaf of $T$ to be split; $\epsilon$ and $\omega$, two thresholds
Ensure: $T$, a new schema profile where $v$ is split
 1: $bestSplit \leftarrow \varnothing$
 2: $rs_v \leftarrow \cup_{s \in S(D_v)} s$            ▷ Set of attributes in the r-schemata of the documents in $v$
 3: for $a \in rs_v$ s.t. $type(a) = \textsf{primitive}$ do                         ▷ Evaluate value-based splits
 4:     $\sigma_a(v) \leftarrow FindBestSplit(a, \epsilon, \omega)$
 5:     if $\sigma_a(v)$ is valid $\wedge\ \sigma_a(v) \prec bestSplit$ then
 6:         $bestSplit \leftarrow \sigma_a(v)$
 7: if $bestSplit = \varnothing$ then                               ▷ If no valid value-based split is found...
 8:     for $a \in rs_v$ do                                             ▷ ...evaluate schema-based splits
 9:         $\sigma_a(v) \leftarrow SchemaBasedSplit(a)$
10:         if $\sigma_a(v) \prec bestSplit$ then
11:             $bestSplit \leftarrow \sigma_a(v)$
12: for $u \in Children(\sigma_a(v))$ do                       ▷ For each leaf $u$ generated by the split...
13:     $AddChild(T, v, u)$                                              ▷ ...add $u$ to $T$ as a child of $v$...
14:     if $|S(D_u)| > 1$ then                                              ▷ ...and, if $u$ is not pure, ...
15:         $T \leftarrow BuildSchemaProfile(T, u)$                                     ▷ ...split it
16: return $T$
```

condition on a (as per Definition 5) and using Definition 9 for split comparison. If no valid value-based split is found for the given threshold ϵ, schema-based splits are tried (lines 8 to 11). Function $SchemaBasedSplit(a)$ returns the schema-based split for attribute a; since all schema-based splits are considered to be valid, one schema-based split is always found at this stage. Finally, T is extended by adding the new leaves generated by the best split found as children of v (lines 12 to 15). If a new leaf u is pure, recursion stops; otherwise, u is recursively split.

7. Experimental Results

The Weka J48 algorithm we use for experimental comparisons requires a dataset structured as a table where each row represents an *observation* (in our case, a document) and each column represents a feature to be used for classification; one additional column is used to store the class each observation belongs to. So we define the *dataset* associated with a collection as follows.

Definition 10 (Dataset). *Let D be a collection. The* dataset *associated with D is a table with the following structure: (i) a column named* rsId*; (ii) one* schema-based column *named* $\textsf{exists\_}path(a)\_type(a)$ *for each attribute* $a \in rs(D)$*; and (iii) one* value-based column *named* $path(a)\_type(a)$ *for each attribute* $a \in rs(D)$ *such that* $type(a) = \textsf{primitive}$. *The dataset includes a row for every* $d \in D$*, such that*

1. rsId *stores a unique identifier given to* $rs(d)$ *within* $S(D)$*;*

2. $\textsf{exists\_}path(a)\_type(a) = 1$ *if* $a \in rs(d)$*, 0 otherwise;*

3. $path(a)\_type(a)$ *stores the value taken by attribute a in document d if* $a \in rs(d)$*,* null *otherwise.*

Table 2: Portion of the dataset for our running example

rsId	ActivityType_primitive	CardioOn_primitive	BPM_primitive	User.Age_primitive	...	exists_ActivityType_primitive	exists_CardioOn_primitive	exists_BPM_primitive	exists_User_object	exists_User.Age_primitive	...
s0	Bike	null	null	null	...	1	0	0	0	0	...
s1	Run	false	null	null	...	1	1	0	0	0	...
s1	Walk	false	null	null	...	1	1	0	0	0	...
s2	Run	true	120	null	...	1	1	1	0	0	...
s3	Run	true	null	null	...	1	1	0	0	0	...
s4	Walk	true	null	42	...	1	1	0	1	1	...
s5	Walk	true	null	65	...	1	1	0	1	1	...

Table 3: Dataset features

Name	*Origin*	*Timespan*	$\lvert D\rvert$	$\lvert S(D)\rvert$	*#columns*	*Depth*	*Opt.*	*Bal.*
SD1	synthetic	—	10 K	8	4 + 7	—	yes	yes
SD2							no	yes
SD3							yes	no
RD1	fitness	4 months	5 M	6	6 + 10	5	yes	no
RD2	fitness	4 months	767 K	139	35 + 38	5	no	no
RD3	sw develop.	27 months	473 K	122	90 + 149	2	no	no

Example 6. *Table 2 shows a portion of the dataset for our running example; for space reasons, some columns are omitted.*

For testing we use a set of both synthetic and real-world datasets, whose characteristics are summarized in Table 3. *Timespan* shows the number of months spanned by real datasets; *#columns* is the number of value-based columns plus the number of schema-based columns in the dataset; *Depth* shows the maximum nesting depth of the JSON documents in real datasets; *Optimal* indicates whether a schema profile exists that is both concise (i.e., with null schema entropy) and explicative (i.e., with no schema-based nodes); *Balanced* indicates whether all the r-schemata have roughly the same number of documents.

Synthetic datasets (i.e., SD1, SD2 and SD3) are relatively small and have been created using a simple rule-based generator: we devised two schema profiles (shown in Figure 5) and then used them as a model to generate the data. The four value-based columns correspond to primitive attributes, shared by all documents (i.e., no value-based column has missing values); the presence/absence of three more object attributes allows eight r-schemata to be distinguished. SD1 is the simplest dataset, where each r-schema has about 1/8 of the 10 000 documents. SD2 presents a balanced but non-optimal situation obtained by assigning, for each r-schema, multiple values to attribute subtype —thus forcing a loss in schema entropy when splitting on that attribute. Lastly, SD3 is gen-

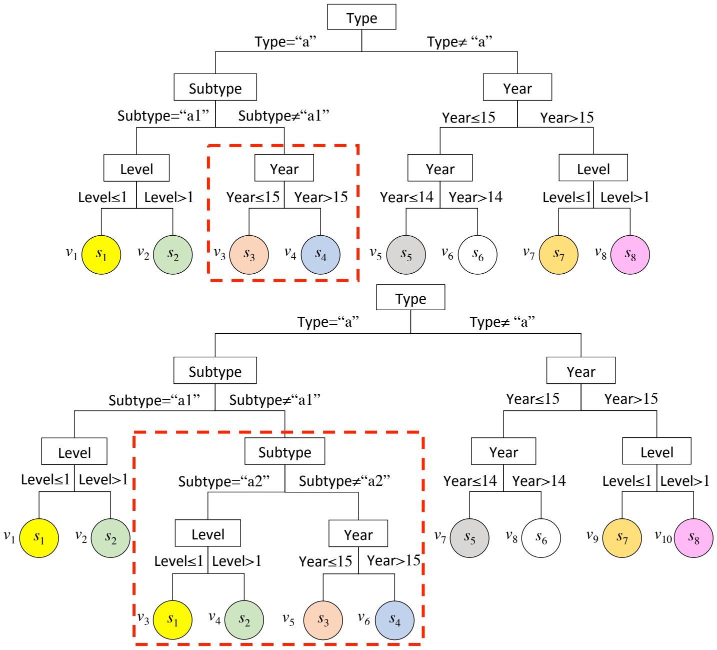


Figure 5: The schema profiles used to generate the synthetic datasets SD1 and SD2 (top), and SD3 (bottom); dashed boxes highlight the differences

erated with the same rules as SD1 but presents an unbalanced situation, where four r-schemata have 9/10 of the documents. All three synthetic datasets can be downloaded at `big.csr.unibo.it/bsp`.

As to real-world datasets, RD1 and RD2 have been acquired from a company that sells fitness equipment, and they contain the registration of new workout sessions and the log of the activities carried out during a workout session, respectively. RD3 has been acquired from a software development company and contains the log of the errors generated by the deployed applications. These three datasets have a key role in evaluating BSP, because they include a large number of documents and present a higher number of attributes and r-schemata.

7.1. Effectiveness

To evaluate the effectiveness of BSP we must verify whether it is really capable of identifying the rules that drive the use of different schemata in documents, which can be done by comparing the schema profiles obtained by BSP with a baseline. To assess the impact of schema entropy and prove that BSP is not a minor improvement over a general-purpose algorithm for building decision trees, the schema profiles are also compared with those obtained from the original version of the Weka J48 algorithm, which only uses entropy. Since schema-based splits typically yield higher gain than value-based ones (and, therefore, are favored by J48), we also run J48 on reduced versions of the datasets that only include value-based columns; we will label with J48-V and J48-VS the schema profiles obtained by ignoring or considering schema-based attributes, respectively. A comparison with other approaches in the literature is not possible since, as shown in Table 1 and discussed in Section 2.1, all those approaches produce a (global, skeleton, or reduced) schema, which by no means can be compared to a schema profile (which has a completely different structure and goal).

To compare two schema profiles we consider the document partition induced by the tree leaves but also the tree structure; more specifically:

- To measure the difference in document partition we rely on the *HR-index* [24], which is a fuzzy variation of the well-known *Rand index* [25] largely used in the literature for comparing partitions. The basic idea behind the Rand index is that two partitions p_1 and p_2 are similar if the pairs of documents that are together (separate) in p_1 are also together (separate) in p_2. The HR-index extends this idea to fuzzy partitions. The value of the index ranges from 0 (completely different partitions) to 1 (identical partitions).

- The *tree edit-distance* (TED) defines the distance between two trees as the length of the cheapest sequence of node-edit operations that transforms one tree into the other [26], and its variants are recognized to be a good choice for comparing XML collections when the structure is particularly relevant [15]. Importantly, TED does not take into account the depth of a node in the tree (i.e., an edit operation on the root costs the same as an edit operation on a leaf). Since this would have a strong impact when comparing decision trees (where the order of decisions is relevant), to measure the difference in tree structure we adopt a modified version of TED which weighs the cost of an edit operation made on a node with the depth of that node; in particular, given a tree of height h and a node v at depth d ($1 \leq d \leq h$), the cost of an edit operation on v is $1/d$. The minimum value for TED is 0 (identical trees), while the maximum value clearly depends on the size of the compared trees.

For synthetic datasets, our testing baseline are the schema profiles used to create the datasets (Figure 5). Table 4 shows the results of the tests made on these datasets to measure the quality of schema profiles with reference to three critical situations (the value of ω is set to zero in these tests, as its manipulation has no remarkable effect on the synthetic datasets due to their small size):

1. *Incomplete information*: in this case the use of different schemata is ruled by either the values of attributes that are not present in the documents or the values of non-primitive attributes for which there is no value-based column in the dataset (e.g., arrays). To assess how the lack of information progressively affects profile building we carried out three tests with an increasing level of incompleteness. We start from the simplest dataset, SD1 (test T1), and progressively remove value-based columns Subtype (test T2)

Table 4: Effectiveness tests on synthetic datasets

Test	*Dataset*	*Algorithm*	ϵ	*Schema entropy*	*#leaves*	*% schema-based nodes*	*TED*	*HR-index*
T1	SD1	J48-V	—	0	8	0%	0	0
		J48-VS	—	0	8	14%	0	0
		BSP	0.1	0	8	0%	0	0
T2	SD1	J48-V	—	1.92	38	0%	9.06	0.061
		J48-VS	—	0	8	29%	1.33	0
		BSP	0.1	0	8	14%	0.5	0
T3	SD1	J48-V	—	2.65	33	0%	9.03	0.097
		J48-VS	—	0	8	42%	1.67	0
		BSP	0.1	0	8	42%	1.17	0
T4	SD2	J48-V	—	0.59	15	0%	3.45	0
		J48-VS	—	0	8	28%	2.32	0.013
		BSP	0.1	0	8	14%	1.48	0.013
			0.3	0.21	10	0%	0	0
T5	SD3	J48-V	—	0.10	12	0%	4	0.001
		J48-VS	—	0	8	14%	2.33	0.125
		BSP	0.1	0	8	0%	0	0

and Level (test T3); threshold ϵ is set at 0.1 for BSP. Obviously, the loss of information in T2 and T3 prevents algorithms from building the exact baseline profile. However, the BSP profiles do not diverge significantly from the baseline, as the split strategy is designed to keep documents with the same r-schema together; thus, BSP resorts to schema-based conditions to recover the lost information and eventually provides the same document partitioning as the baseline.

2. *Non-optimal dataset*: in SD2, the use of different schemata is ruled by complex conditions that are not considered by our search strategy; hence, no concise and explicative schema profile can be found and one of these two features must be sacrificed. Test T4 shows how this trade-off can be achieved by adjusting ϵ. With $\epsilon = 0.1$, BSP is less tolerant to the separation of documents of the same r-schemata into different leaves; this results in BSP preferring schema-based conditions and generating a concise (8 leaves and zero schema entropy) but inaccurate (high TED and non-zero HR-index) schema profile. The baseline actually splits the documents of r-schemata s_1 and s_2 into 4 leaves (see Figure 5), and this result can be achieved by increasing ϵ to 0.3, at the expense of a substantial loss in terms of schema entropy.

3. *Unbalanced dataset*: in this case the documents are unevenly distributed among the r-schemata. We run a single test (T5) on the unbalanced dataset SD3, with threshold ϵ still set at 0.1. The results show that

only BSP is capable of perfectly reproducing the baseline, as the schema entropy measure favors a pure separation of the r-schemata even if the entropy gain is limited (due to the unbalanced distribution).

The comparison with J48-V and J48-VS helps us outlining the strengths of BSP that eventually yield better results than a standard decision tree algorithm. In particular, we observe that — except for T1 — J48-V builds more complicated schema profiles with a higher number of leaves. This is due to the fact that entropy alone does not encourage conciseness; rather, it tends to maximize the gain by separating the documents belonging to the most frequent r-schemata, even if the documents of the less frequent r-schemata end up to be split in multiple leaves. Eventually, the schema profiles of J48-V often result in a document partitioning that diverges from the one of the baseline. Additionally, the schema profiles built by J48-VS show the importance of considering the difference between value- and schema-based columns. These profiles tend to overuse schema-based conditions, even when there is no issue of incompleteness (i.e., T1, T4, and T5), thus failing to identify value-based splits that would have been more explicative and more accurate.

For our real-world datasets a complete baseline is not available; indeed, during requirement elicitation our users could only provide a very small schema profile consisting of the two or three value-based conditions that they presumed to be representative of the schema usage. Given such premise, we start by studying the behavior of BSP with different values of ϵ and ω. The three diagrams in Figure 6 show the effects of the two thresholds on the schema profile for RD2 in terms of conciseness (given by the number of leaves and by the schema entropy) and its explicativeness (given by the percentage of value-based nodes). The first observation is that, when a very concise schema profile is requested by the user (i.e., $\epsilon = 0$ and the schema entropy is therefore forced to be null), the major risk is to renounce to explicativeness (as shown by the low percentage of value-based nodes). As ϵ is increased, the general tendency is to create more explicative schema profiles. However, by allowing splits that increase the schema entropy, BSP may be inclined to choose those splits that single out only a small fraction of the documents (i.e., splits yielding a very low loss, but also a very low gain); in this case, the risk is to obtain a very complex schema profile. Conversely, the increase of ω favors conciseness by filtering out the aforementioned splits.

As the order of the conditions is very relevant, users may be interested more in the upper parts of the schema profile than in the lower ones. So, with the next test we analyze the structure of the schema profile by segmenting the tree at different levels of depth. Figure 7 shows the level-by-level decomposition of the schema profile generated on RD2. Based on the results of the previous test, we set $\epsilon = 0.01$ and $\omega = 0.1$. The figure shows that the entropy rapidly decreases, while the low schema entropy ensures the conciseness of the schema profiles across every stage. Most importantly, the percentage of schema-based nodes is kept quite low within the first levels, denoting a good degree of explicativeness of the upper part of the schema profile. Schema-based conditions tend to increase

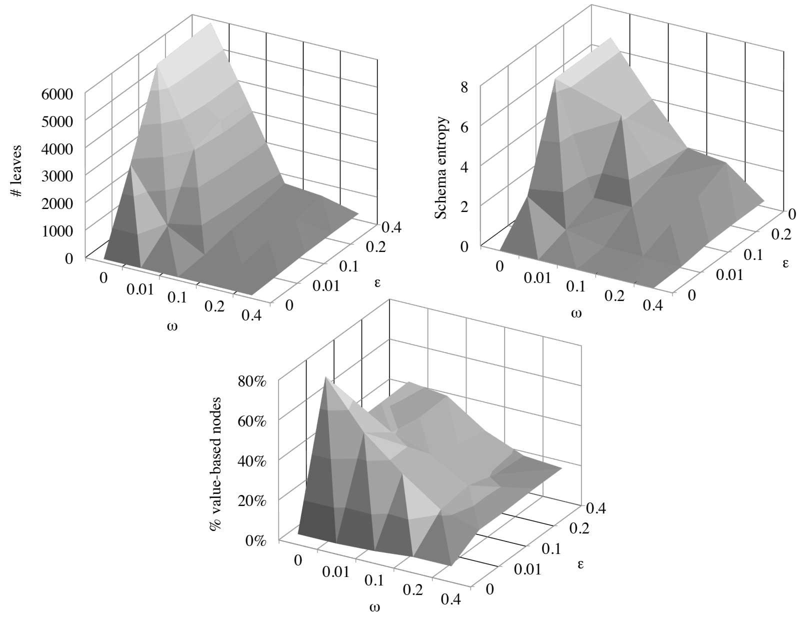


Figure 6: The effects of different values of ϵ and ω on RD2

in the lower parts, where a good degree of separation of the r-schemata has already been achieved (as proven by the low entropy).

As a final test, we evaluate the quality of the schema profiles generated for real-world datasets with respect to the baseline provided by the user. Since the baseline is limited to two or three value-based conditions, a comparison against complete schema profiles would inevitably be unfair and reveal great dissimilarities. Therefore, the values of TED have been determined by limiting each schema profile to a number of levels that equals the one in the baseline; the calculus of the HR-index is omitted for the same reason. The values of ϵ and ω for RD1 and RD3 have been determined by a statistical evaluation, as previously shown for RD2. Table 5 shows the results of the test. In particualr, BSP always provides a very concise schema profile, which also matches with the baseline in RD1. The comparison with J48-V and J48-VS confirms the issues that would emerge by adopting the basic algorithms — i.e., a less accurate and over-complicated schema profile by relying on entropy alone, and an overuse of schema-based columns without differentiating between value- and schema-based conditions.

7.2. Efficiency

The driving factor in estimating the complexity for computing the gain and loss of a split is the number of documents, $|D|$ (since the number of r-schemata, $|S(D)|$, and the number of leaves, m, are clearly quite lower than $|D|$). Under this assumption, we know from the literature that the complexity for evaluating

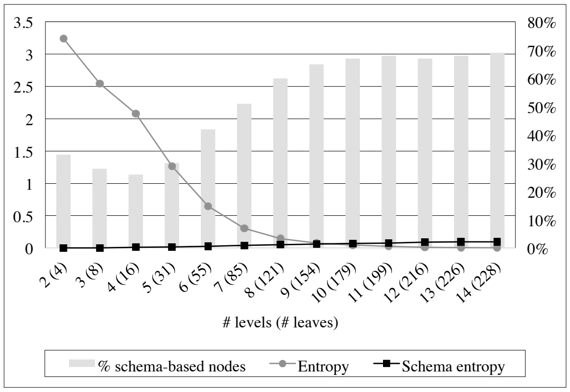


Figure 7: Level-by-level decomposition of the schema profile generated on RD2 with $\epsilon = 0.01$ and $\omega = 0.1$

all the possible splits of leaf v on attribute a by computing their gain and loss is $O(|D_v|)$ if a is categorical (including the case in which a is a schema-based columns), $O(|D_v| \cdot (log|D_v|+1))$ if a is numeric [27]. Within an iteration of BSP, this cost is paid whenever functions $FindBestSplit$ and $SchemaBasedSplit$ are called. In turn, the number of calls depends on the number of attributes belonging to rs_v.

Formally modeling the complexity of the basic operations is not sufficient to evaluate the overall BSP execution time. The number of recursive calls is related to the effectiveness of the purity measures, to the availability of useful value-based attributes, and to the inherent complexity of the dataset. To properly analyze this holistic bundle of factors we compare the BSP execution time with the ones of J48-V and J48-VS on real-world datasets[8]; the tests were made on a 64-bits Intel Core i7 quad-core 3.4GHz, with 16GB RAM, running Windows 7 pro SP1. For the same tests of Table 5, Table 6 shows:

- the total execution times (in seconds) to build the schema profiles;
- the number of times that functions $FindBestSplit$ and $SchemaBasedSplit$ are called (shown as *#FBS* and *#SBS*, respectively);
- the average number of documents involved when functions $FindBestSplit$ and $SchemaBasedSplit$ are called (shown as *Avg #docs per FBS* and *Avg #docs per SBS*, respectively);

Note that (i) the number of iterations does not necessarily reflect the size of the tree, because J48 adopts post-pruning techniques that eventually reduce the size of the schema profile; (ii) the time for post-pruning is not considered because of its irrelevance (not greater than a second); (iii) BSP is not penalized by the

[8] Synthetic datasets are not considered for evaluating efficiency since their execution time is below one second.

Table 5: Effectiveness tests on real-world datasets

Dataset	*Algorithm*	ϵ	ω	*Schema entropy*	*Height*	*#leaves*	*% schema-based nodes*	*TED*
RD1	J48-V	-	-	2.16	10	48	0%	0
	J48-VS	-	-	0	3	6	60%	1
	BSP	0.01	0.1	0	3	6	80%	0
RD2	J48-V	-	-	3.69	35	521	0%	2.33
	J48-VS	-	-	0	17	115	93%	1.33
	BSP	0.01	0.1	0.10	14	228	69%	1.33
RD3	J48-V	-	-	2.07	34	243	0%	1
	J48-VS	-	-	0	12	85	73%	1
	BSP	0.01	0.01	0.10	26	231	50%	0.5

added calculation of schema entropy, since this requires the same data used to calculate the entropy (as per Definition 7).

By looking at the results, we observe that the execution time of BSP is consistent with the one of J48. More specifically, BSP is always faster than J48-VS; this is due to the fact that BSP does not try schema-based splits if a valid value-based one has been found. This is confirmed in RD1 and RD3 by the lower value of #SBS, and in RD2 by the lower number of involved documents (because BSP tends to resort to schema-based columns only in the lower levels of the schema profile). With respect to J48-V, we observe that the ability of BSP to quickly converge to a precise schema profile requires a much lower number of calls to the FBS function. However, this does not necessarily correspond to a boost in performance due to the higher complexity of the FBS function in BSP to evaluate schema-based splits.

8. Conclusions

In this paper we have presented BSP, an approach to schema profiling for document-oriented databases. To the best of our knowledge, BSP is the first approach to schema profiling based on extensional information. The idea is to capture the rules that explain the use of different schemata within a collection through a decision tree whose nodes express either value-based or schema-based conditions. Guided by the requirements elicited from users, we have proposed an algorithm that builds precise, concise, and explicative schema profiles. The experimental tests have shown that BSP is capable of achieving a good trade-off among these features and of delivering accurate schema profiles.

Our future work in this field will develop along two different perspectives. On the implementation side, we will incorporate user interaction in the schema profile building algorithm to give users a closer control on the trade-off among

Table 6: Efficiency tests on real-world datasets

Dataset	*Algorithm*	ϵ	ω	*Time (sec.)*	*#FBS*	*#SBS*	*Avg #docs per FBS*	*Avg #docs per SBS*
RD1	J48-V	-	-	172	1270	-	225 K	-
	J48-VS	-	-	43	27	12	2839 K	3290 K
	BSP	0.01	0.1	42	26	11	3320 K	3162 K
RD2	J48-V	-	-	125	7515	-	27 K	-
	J48-VS	-	-	139	2784	550	59 K	184 K
	BSP	0.01	0.1	116	2851	602	57 K	91 K
RD3	J48-V	-	-	156	4282	-	36 K	-
	J48-VS	-	-	229	991	1095	124 K	138 K
	BSP	0.01	0.01	191	1984	558	64 K	39 K

the different features of schema profiles and to inject additional knowledge in it. Also, we will evaluate the state of the art of decision tree algorithms to enhance the performance of BSP (e.g., by incrementally building the decision tree [28, 29, 30], by introducing approximation mechanisms [31, 32], or by deploying the algorithm on a big data infrastructure [33]). From a more research-oriented point of view, we will investigate how to take advantage of schema profiles when querying document collections; for instance, we will use schema profiles to enhance schema-on-read approaches to analytical querying in business intelligence contexts [1, 2].

References

[1] Z. H. Liu, D. Gawlick, Management of flexible schema data in RDBMSs - opportunities and limitations for NoSQL, in: Proc. CIDR.

[2] X. L. Dong, D. Srivastava, Big data integration, in: Proc. ICDE, pp. 1245–1248.

[3] S. Nestorov, J. Ullman, J. Wiener, S. Chawathe, Representative objects: Concise representations of semistructured, hierarchical data, in: Proc. ICDE, pp. 79–90.

[4] Q. Y. Wang, J. X. Yu, K.-F. Wong, Approximate graph schema extraction for semi-structured data, in: Proc. EDBT, pp. 302–316.

[5] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, XTRACT: a system for extracting document type descriptors from XML documents, SIGMOD Record 29 (2000) 165–176.

[6] J. Hegewald, F. Naumann, M. Weis, XStruct: Efficient schema extraction from multiple and large XML documents, in: Proc. ICDE Workshops, pp. 81–81.

[7] G. J. Bex, W. Gelade, F. Neven, S. Vansummeren, Learning deterministic regular expressions for the inference of schemas from XML data, ACM TWEB 4 (2010) 14.

[8] J. L. C. Izquierdo, J. Cabot, Discovering implicit schemas in JSON data, in: Proc. ICWE, pp. 68–83.

[9] M. Klettke, U. Störl, S. Scherzinger, O. Regensburg, Schema extraction and structural outlier detection for JSON-based NoSQL data stores., in: Proc. BTW, volume 2105, pp. 425–444.

[10] D. S. Ruiz, S. F. Morales, J. G. Molina, Inferring versioned schemas from NoSQL databases and its applications, in: Proc. ER, pp. 467–480.

[11] L. Wang, S. Zhang, J. Shi, L. Jiao, O. Hassanzadeh, J. Zou, C. Wangz, Schema management for document stores, Proc. VLDB Endowment 8 (2015) 922–933.

[12] P. A. Bernstein, J. Madhavan, E. Rahm, Generic schema matching, ten years later, Proc. VLDB Endowment 4 (2011) 695–701.

[13] R. Nayak, W. Iryadi, XML schema clustering with semantic and hierarchical similarity measures, Knowledge-Based Systems 20 (2007) 336–349.

[14] M. L. Lee, L. H. Yang, W. Hsu, X. Yang, XClust: clustering XML schemas for effective integration, in: Proc. CIKM, pp. 292–299.

[15] G. Guerrini, M. Mesiti, I. Sanz, An overview of similarity measures for clustering XML documents, in: Web Data Management Practices: Emerging Techniques and Technologies, Idea Group, 2007, pp. 56–78.

[16] P. Bohannon, E. Elnahrawy, W. Fan, M. Flaster, Putting context into schema matching, in: Proc. VLDB, pp. 307–318.

[17] H. Nottelmann, U. Straccia, sPLMap: A probabilistic approach to schema matching, in: Proc. ECIR, pp. 81–95.

[18] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, P. Domingos, iMAP: discovering complex semantic matches between database schemas, in: Proc. ICMD, pp. 383–394.

[19] J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.

[20] C. E. Shannon, A mathematical theory of communication, ACM SIGMOBILE Mobile Computing and Communications Review 5 (2001) 3–55.

[21] H. Deng, G. Runger, E. Tuv, Bias of importance measures for multi-valued attributes and solutions, in: Proc. Int. Conf. on Artificial Neural Networks, pp. 293–300.

[22] J. R. Quinlan, Induction of decision trees, Machine learning 1 (1986) 81–106.

[23] S. R. Safavian, D. Landgrebe, A survey of decision tree classifier methodology, IEEE TSMC 21 (1990) 660–674.

[24] E. Hullermeier, M. Rifqi, A fuzzy variant of the rand index for comparing clustering structures, in: Proc. IFSA-EUSFLAT, pp. 1294–1298.

[25] W. M. Rand, Objective criteria for the evaluation of clustering methods, Journ. of the American Statistical association 66 (1971) 846–850.

[26] S. M. Selkow, The tree-to-tree editing problem, Information processing letters 6 (1977) 184–186.

[27] S. Ruggieri, Efficient c4. 5 [classification algorithm], IEEE TKDE 14 (2002) 438–444.

[28] P. Utgoff, Incremental induction of decision trees, Machine learning 4 (1989) 161–186.

[29] S. Crawford, Extensions to the CART algorithm, Int. Jour. of Man-Machine Studies 31 (1989) 197–217.

[30] D. Kalles, T. Morris, Efficient incremental induction of decision trees, Machine Learning 24 (1996) 231–242.

[31] J. Gehrke, V. Ganti, R. Ramakrishnan, W.-Y. Loh, BOAT-optimistic decision tree construction, SIGMOD Record 28 (1999) 169–180.

[32] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proc. SIGKDD, pp. 71–80.

[33] W. Dai, W. Ji, A MapReduce implementation of C4.5 decision tree algorithm, Int. Jour. of Database Theory and Application 7 (2014) 49–60.