Rodrigues and Silva *Journal of the Brazilian Computer Society* (2021) 27:14
https://doi.org/10.1186/s13173-021-00119-5

Journal of the
Brazilian Computer Society

RESEARCH Open Access

A study on machine learning techniques for the schema matching network problem

Diego Rodrigues and Altigran da Silva*

*Correspondence:
alti@icomp.ufam.edu.br
Instituto de Computação,
Universidade Federal do Amazonas,
Manaus, Brazil

Abstract

Schema matching is the problem of finding semantic correspondences between elements from different schemas. This is a challenging problem since disparate elements in the schemas often represent the same concept. Traditional instances of this problem involved a pair of schemas. However, recently, there has been an increasing interest in matching several related schemas at once, a problem known as *schema matching networks*. The goal is to identify elements from several schemas that correspond to a single concept. We propose a family of methods for schema matching networks based on machine learning, which proved to be a competitive alternative for the traditional matching problem in several domains. To overcome the issue of requiring a large amount of training data, we also propose a bootstrapping procedure to generate training data automatically. In addition, we leverage constraints that arise in network scenarios to improve the quality of this data. We also study a strategy for receiving user feedback to assert some of the matchings generated and, relying on this feedback, improve the final result's quality. Our experiments show that our methods can outperform baselines, reaching F1-score up to 0.83.

Introduction

Schema matching is the task of finding semantic correspondences between elements (or attributes) of two given database schemas [1–5]. Such a task is essential for enabling data integration and systems interoperability in domains such as e-commerce, geospace, biology, health, etc.

The schema matching task is challenging for many reasons. First, schema elements, e.g., attributes representing the same concept, may have different names in different schemas. On the other hand, elements with similar names may refer to distinct concepts. Also, equivalent elements in two schemas may have a different structure. Finally, there may be the case in which many elements from one schema represent a concept represented by a single element in the other schema.

Consider the (simplified) schemas of databases on academic information from universities illustrated in Fig. 1. The schema matching task is to identify the matchings (depicted as dotted lines) between elements from those schemas. In this example, the

diagram on the left illustrates how the database schema from University A (UNIA) models data about students. The diagram on the right illustrates how the database schema from University B (UNIB) models the same data. Notice that the same data is represented differently, even using particular representations of hierarchy and attribute names. For instance, `UNIA.PERSONALDATA.NAME.Firstname` and `UNIB.Biography.fname` represent the student's first name. Also, there might be more challenging matchings to be discovered, e.g., `UNIA.BIRTHINFORMATION.Dateofbirth` and `UNIB.Biography.dob`, both are representing the student's date of birth, which may be validated only by a domain specialist or the schema owners. Finally, some elements might not have any correspondences on the other schema, e.g., `UNIB.Biography.citizenship`.

Traditionally, the schema matching task is performed manually by specialists with extensive knowledge about the schemas and their domain. However, even for a specialist, this task may be time-consuming, costly, and error-prone. Over the years, several research initiatives have been carried out to deal with schema matching, resulting in several papers published [2, 3, 6–9] and several prototypes and commercial systems made available [10–12]. Many of these methods rely on a set of predefined steps and parameters [2, 11, 13]. Others rely on machine learning to create specific models for every matching task [4, 14, 15]. Heuristics can perform well in certain domains; however, they often need tuning for achieving good results. On the other hand, machine learning methods can adapt to different matching tasks once a substantial amount of training data is available, which may be hard to obtain.

As the problem evolved, schema matching tasks have appeared in settings where more than two data sources (e.g., database, query forms) need to be matched [9, 16–20], as if they compose a *network* of schemas. The task that involves more than two schemas is known as the *schema matching network* task. Figure 2 shows an example of a schema matching network. As in Fig. 1, we have (simplified) schemas of databases on academic information from universities, where schema elements representing the same concept are linked with dotted arrows. However, we now have a schema matching network with four schemas, in which elements might have correspondences in one or more schemas.

The schema matching network task essentially shares the same characteristics of the traditional pairwise setting. However, there are some new challenges, as well as opportunities. The more obvious challenge is the scale. As more schemas are involved, more schema elements will be matched, and the number of possible matching combinations to

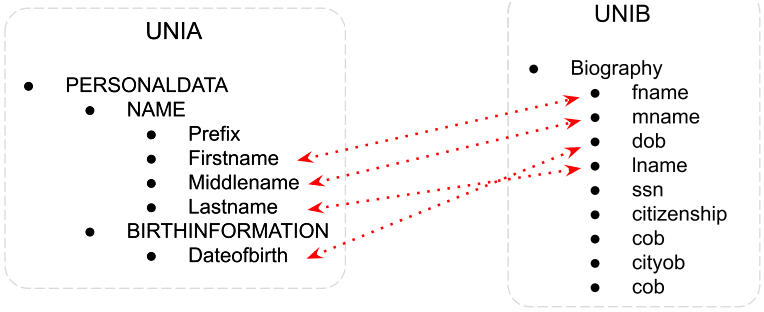


Fig. 1 Matching elements from two simplified databases schemas

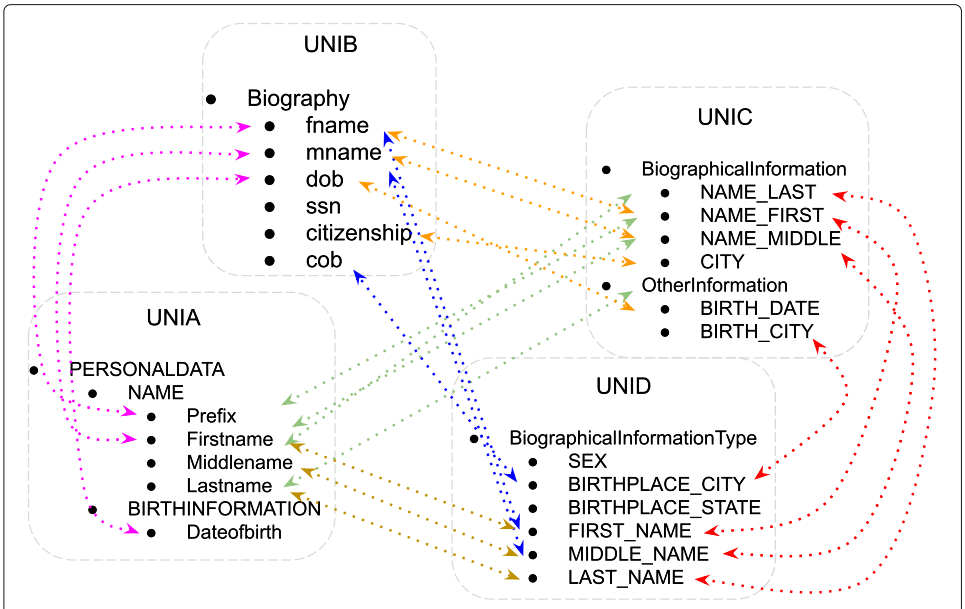


Fig. 2 A schema matching network: four schemas on the universities domain

evaluate may grow exponentially. In addition, there is also the challenge of guaranteeing the consistency of the matches across the whole network; that is, if some element e_0 from a schema S_0 matches elements $e_1, \ldots, e_n$ from schemas $S_1, \ldots, S_n$, respectively, for a matter of consistency, it is reasonable to expect that $e_1, \ldots, e_n$ may match consistently among themselves. On the other hand, and despite these challenges, there are also opportunities. Based on the consistency requirements, one can define network-level integrity constraints (e.g., [17–19]) and rely on them to verify the correctness of the matching process. Also, any knowledge discovered during the matching process can be shared instantaneously across the whole network as proposed by Madhavan et al. [17].

Since machine learning methods have been applied to the pairwise scenario with good results, in our work, we experiment with these methods for schema matching networks. This brings, however, additional challenges, notably the necessity of a larger volume of labeled data and dealing with unbalanced datasets, in which the number of non-matching of pairs is much higher than the number of matching pairs. To address these issues, we study several techniques: using black-box schema matching systems to generate training examples, relying on network constraints to build good and informative training sets, and leveraging evaluations from users to help to improve the final matching quality, avoiding making it a laborious task.

In a nutshell, our contributions are as follows. First, we evaluate some popular machine learning methods previously used for the schema matching problem. Our study aims to verify if machine learning approaches are suitable to the schema matching network problem treated as a classification problem. It enables us to choose one machine learning method to act as a base learner. Next, we study if network constraints can help improve the training phase and, consequently, matching results. Finally, we study the reconciliation task, where users may review, validate, correct the results, and show that user input

can help improve even more the matching results of a method. We also reflect on the different ways and stages to request the user input and how much this can change the quality of matching results.

The remainder of this paper is organized as follows: In the "Background and related work" section, we present an overview of the schema matching problem and how methods in the literature have tackled this problem. In the "Using machine learning in schema matching networks" section, we discuss how to train classifiers to perform the schema matching network task. Then, we present a method to leverage using heuristic methods and integrity constraints to obtain automatically labeled training examples. Finally, we present a follow-up method to perform the *reconciliation* task, in which the method uses network constraints and user inputs to improve the matching network quality. In the "Experimental evaluation" section, we present experiments performed to evaluate our methods. We show that our method can train a classifier to achieve up to 0.89 in terms of precision. Next, we show that our automatically trained method reaches, on average, an F1-score of 0.74, topping representative baselines. Finally, we show that the matching quality can be improved, on average, 13% and using at least 4 times less user input than other methods. In the "Conclusions and future work" section, we provide some remarks and conclusions on our work.

Background and related work

The schema matching problem has been studied for a long time in the literature, with several surveys and books covering this topic deeply and widely [5, 6, 21, 22]. In this section, we only highlight work close to our study.

Classic schema matching

Schema matching is the task of finding semantic correspondences between elements of two schemas [2]. The schemas can be from any distinct heterogeneous data sources (e.g., database schemas, XML DTDs, HTML form tags, etc.) known to be in the same domain [23]. This task is one of the first steps in any data integration process, as it results in connecting two different data sources [6].

Despite several prototypes and methods being presented over the years addressing this task, no current method is considered as having completely solved the problem. Also, to guarantee the quality of matching results, a specialist user is often required to review the answers after executing a method.

Usually, schema matching methods apply one or more functions to establish a similarity value between pairs of elements from the schemas. Each pair is called a *matching candidate*. These functions, called *matchers*, receive two elements as input and estimate a similarity value between 0 and 1. The higher the value, the more similar the elements are. Matchers can use several different strategies to estimate similarities, for instance, comparing schema element names, their semantic similarity using a thesaurus, their data types, cardinality, or even data values if available.

Table 1 shows an example of similarity values generated by the well-known Levenshtein distance function [24, 25][1] calculated for some elements from two datasets in

[1] The Levenshtein distance is a lexic similarity between two strings. Given two strings s and t, their similarity is the ratio of their edit distance by the size of the bigger string, $similarity(s,t) = \frac{edit\_distance(s,t)}{max(len(s),len(t))}$

Table 1 Similarity values based on the Levenshtein distance

Levenshtein matcher		Schema B				
		searchForm .search	**searchForm .searchType**	**searchForm .keyword**	**searchForm .shortTitle**	**searchForm .shortAuthor**
Schema A	search	1.0000	0.6000	0.1429	0.2000	0.2727
	search.field-title	0.0909	0.1818	0.0909	0.3636	0.0000
	search.field-subject	0.1538	0.1538	0.0769	0.0769	0.0769
	search.field-asin	0.2000	0.0000	0.1000	0.0000	0.0000
	search.field-publisher	0.1333	0.1333	0.0667	0.1333	0.2000
	search.field-dateyear	0.1429	0.1429	0.1429	0.1419	0.1429
	search.field-keywords	0.1429	0.1429	0.5000	0.0000	0.1429

the Books domain. In this example, if a method chooses to apply a threshold value of 0.7 in that similarity matrix, the only matching candidate that would be selected is ⟨`search`,`searchForm.search`⟩. As another example, a method could rank all similarities and select the top-k values from the matrix. For k=2, this strategy would allow selecting matchings ⟨`search.field-keywords`,`searchForm.keyword`⟩ and ⟨`search.field-title`,`searchForm.shortTitle`⟩.

Methods can choose to apply heuristics to combine matchers and select matching candidates. Other methods rely on machine learning techniques to create models to predict if a matching candidate is a true matching or not. Alternatively, methods can query a specialist to obtain correct matchings.

Heuristic methods

Systems such as COMA [2], Similarity Flooding [7], CUPID [3], and Agreement Maker [13] are representative systems that use heuristics for combining matchers.

COMA/COMA++ [2, 11], whose name is an acronym for "*CO*mbining *M*atching *A*lgorithms," is a method that uses several distinct algorithms that implement similarity functions and combines them to generate matchings between two given schemas. Its execution is a good example of how heuristic methods work. Its flow of execution is as follows: (a) it starts by receiving two input schemas from the same domain; (b) all pairs of elements from the schemas are submitted to pairwise functions called *matchers* (such as the *Levenshtein distance*). Each function calculates a similarity value ranging between $[0, 1]$, being 1 the similarity score that indicates a high similarity. Each matcher generates a similarity matrix, such as the one shown in Table 1. As a result, every pair of elements has a similarity value assigned by each matcher. After calculating the similarities, an aggregator function (such as the *Average*) is used to summarize all the similarities calculated by the matchers into a single similarity matrix; (c) finally, the system applies a selection method, such as using *thresholds* or selecting the top-k matches, to generate the method's matching answers and (d) presents them to the user.

The authors report that the best combinations of strategies are mainly lexical functions. In fact, they indicate that a subset of the matchers proposed leads to the best combination of matchers. They also tested different combinations of aggregation and selection functions. The experiments were performed on ten pairwise matching tasks from the *Purchase Order* domain. The authors report the results for different configurations of COMA and argue that each different configuration can be used in different scenarios. When its

best configuration (also referred to as the default configuration) is used, COMA achieved Overall[2] (a combination of precision and recall) values ranging from 0.6 to 0.7.

Although COMA has matchers that consider the structural hierarchy of elements, their similarities are diluted by the aggregation function, losing this important aspect when addressing the schema matching problem. Similarity Flooding [7] appears as an alternative that strongly considers the structural aspect of the schemas, as its algorithm is based on graph analysis.

The Similarity Flooding algorithm starts by transforming the two input schemas into graphs. Then, it applies a string matcher to estimate initial similarities between pairs of elements from the schemas. Next, the flooding algorithm propagates similarities through graph nodes from top to bottom. Similarities between elements in the same branches receive partial scores from ancestor elements. Finally, the algorithm applies a threshold to prune the less plausible matchings, and the highest similarities are given as the method's matching answers. The authors report experiments using nine pairwise matching tasks whose ground truth was provided by volunteers. On average, the method reached an accuracy value of around 0.55. The authors note that the algorithm could perform better in some of the tasks, as the schemas had a higher amount of structural information.

Among other relevant heuristic methods for schema matching, we may cite CUPID [3], Agreement Maker [13], and the work by Shiang et al. [26].

The heuristic methods are largely used as they are simple to implement and execute. They often run fast and generate a large number of correct matches. However, they are not always consistent when running in different datasets. Previous studies [5, 21] observed that these methods can perform better depending on the dataset and the chosen parameters. Some systems such as eTuner [27] and SMB [28] were devoted to determining how tuning parameters can improve the matching.

Machine learning methods

Some methods in the literature approach the matching problem as a classification problem, where a machine learning model has to decide if a matching candidate is a `TRUE` or `FALSE` matching, depending on whether or not they represent the same concept. According to such an approach, a schema matching task consists of two schemas $\mathcal{S}_0$ and $\mathcal{S}_1$ that should be matched together. Then, there is a set $C = \{c_1, c_2, ..., c_j\}$ of matching candidates, where each candidate $c \in C$ consists of two schema elements s and t, each from a different schema, a vector v of similarity values between them, given by several matchers, which can be viewed as features of the matching candidate, and a label l, which should have the value `TRUE` if s and t form a true pair of corresponding elements, or `FALSE` otherwise. The ultimate goal is to find all the matching candidates in C that have the label $l =$ `TRUE`.

Recalling the example from Table 1, we can build a set of candidate matches by taking all possible combinations of elements in Schema A and Schema B, such as ⟨`search`, `searchForm.search`⟩, then ⟨`search`, `searchForm.searchType`⟩ and so on. We see in the example that the Levenshtein similarity for the pair ⟨`search`,`searchForm.search`⟩ is 1.00. Other matchers can also be used. Consider that a data type matcher has also evaluated the pair with value 1.00 as both elements take `string` values. Hence, the similarity vector of the candidate would be $v = [1.00, 1.00]$.

[2] Overall is defined by the authors as $recall * (2 - \frac{1}{precision})$.

Finally, if this candidate is considered a true pair of corresponding elements, its label value is $l = \texttt{TRUE}$.

State-of-the-art methods use several learning strategies, such as combining several decision trees to building a meta-model [14, 29], combining different learning algorithms to create more complex models [28, 30], and using specialists inputs to help to learn models to weigh strategies better and to tune learning parameters [8].

A few methods in the literature have applied machine learning techniques to the problem of combining matchers. Among the most representative methods in this category, we cite LSD [30], SMDD [8], YAM [14], and SMB [28].

LSD [30] evaluates element-level matchers using previously obtained training data. Then, a meta-learner selects and weighs matchers that classify matching candidates in new matching tasks. YAM/YAM++ (Yet Another Matcher) [14, 31] uses a similar approach: it trains several learning methods using training data. The models are evaluated by using previously labeled examples. The best model generated is used to generate correspondences to new matching tasks. ALMa (*A*ctive *L*earning *Ma*tching) [29] uses an ensemble of decision trees to decide matchings. When trees cannot generate a decision, it queries the user for clarification. The method takes the user answers to re-train models and re-do the process.

Other methods we notice are SMDD—Schema Matching based on Data Distribution [8], which uses neural networks; SMB [28], which uses the AdaBoost method; and the work by Carvalho et al. [15], which proposes a genetic programming approach to find complex matches.

Since learning classifiers require a training set, the user must often classify a large number of instances, which may be hard to carry out. This requirement is a disadvantage in comparison to the previously described heuristic methods, which are unsupervised. On the other hand, machine learning-based methods can generate different models for different matching tasks, leading to potentially better results.

Schema matching networks

With the number of sources getting higher and the growing necessity of making systems interoperate, creating *global schemas* to represent many different sources in an integration process has been a necessity. However, when adding new sources to the process, the global schema created often needs to be rebuilt, which is hardly a practical solution [32]. Moreover, applications are getting more and more complex and require more than two schemas, unlike the classic schema matching problem. Schema matching network takes as input a (potentially large) set of schemas in the same domain and outputs matches of elements of the schemas altogether [33].

We illustrated in Fig. 2 an example of a schema matching network involving four schemas on the universities application domain.

A schema matching network task is a tuple $\mathcal{N} = \langle \mathcal{S}, C, \Omega \rangle$, where $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ is a collection of schemas that should be matched together, $C = \{c_1, c_2, ..., c_j\}$ is a set of matching candidates, and $\Omega = \{\omega_1, \omega_2, ...\}$ is a set of network constraints. For each pair of elements $\langle s, t \rangle \in S_i \times S_j, \forall S_i, S_j \in \mathcal{S}, i \neq j$, there is a candidate match $c \in C$ of the form $c = \langle s, t, l, v \rangle$. As in the classic schema matching setting, v is a vector of similarity values and l is a label whose value is `TRUE` if s and t form a true pair of corresponding schema elements or `FALSE` otherwise. In a typical schema matching network task $\mathcal{N}$, the goal is

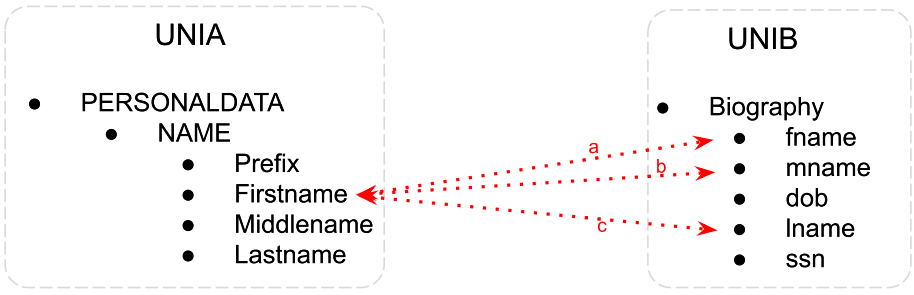


Fig. 3 One-to-one constraint violation

to find all matching candidates c which the label value l is `TRUE`. The correspondences in the matching network should comply with the set of network constraints Ω. In this work, we consider two network constraints:

One-to-one constraint. Any attribute of a schema has at most one matching attribute in another schema. In Fig. 3, a, b, and c are uncertain matchings as the attribute `Firstname` from UNIA has three correspondences in the schema from UNIB. Formally, ω_1 (*one-to-one constraint*) is defined as follows: Consider elements $a \in S_1$ and $b \in S_2$. If a matching $c = \langle a, b \rangle$ has label $l =$ `TRUE`, no other candidate $c_x = \langle a, x \rangle (x \in S_2 \wedge x \neq b)$ can have $l_x =$ `TRUE` and no other candidate $c_y = \langle y, b \rangle (y \in S_1 \wedge y \neq a)$ can have $l_y =$ `TRUE`.

Cycle constraint: Attributes matched together in different schemas must form a cycle. In Fig. 4, the matchings $\langle k, l, m \rangle$ violate the cycle constraint as they do not close a cycle. To fix this inconsistency, either matching k should be changed to connect `Middlename`(UNIA) and `mname`(UNIB) or matching m should be changed to connect `fname`(UNIA) and `NAME_MIDDLE`(UNIC). Formally, ω_2 (*cycle constraint*) is defined as follows: Consider elements $a \in S_1$, $b \in S_2$ and $c \in S_3$. If the matching candidates $c_x = \langle a, b \rangle$ and $c_y = \langle b, c \rangle$ have labels $l_x, l_y =$ `TRUE`, then the candidate $c_z = \langle a, c \rangle$ must have the label $l_z =$ `TRUE`.

When automatic methods generate a set of matchings, these matchings are often inconsistent with each other in the sense that they violate the constraints above, resulting in an *uncertain* matching network. The inconsistent matchings are referred to as *uncertain*

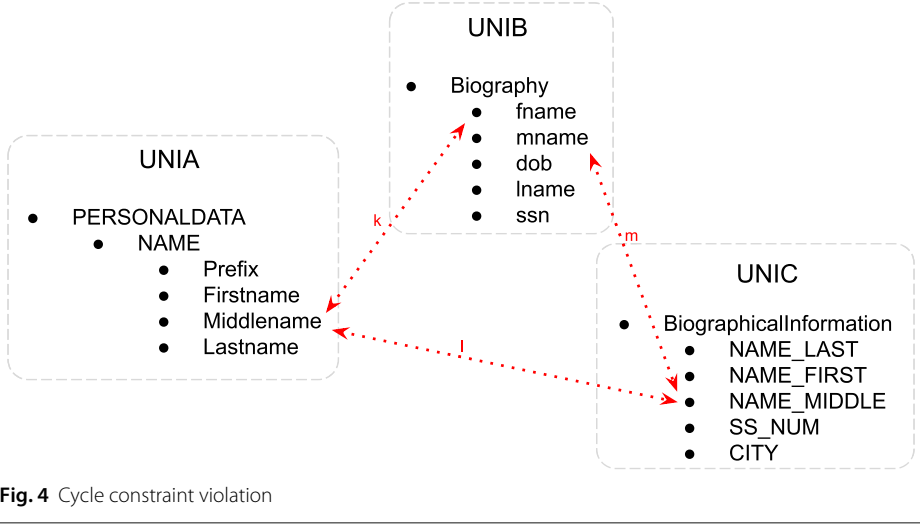


Fig. 4 Cycle constraint violation

matchings. To help detect such inconsistencies, some methods rely on heuristics inspired in studies on mapping messages between P2P systems [32, 34]. The constraints used in our work are domain-independent, and other constraints may be added depending on the matching task if needed.

Some methods [16–19] address the schema matching network problem when a global schema is known and can be matched against the other schemas. The MGS framework [16] aims at matching input schemas by finding correspondences between them and a predefined schema. Corpus-based schema matching [17] builds a corpus of schemas in a given domain to improve matching algorithms. The corpus is a collection of schemas and mappings between some schema elements. Schemas in the corpus are related but need not be mapped to each other. Since different designers defined the schemas, the corpus has several representations of each concept in the domain. When new schemas are matched, their elements are matched against the set of concepts. Holistic schema matching (HSM) [18] uses a similar strategy but focuses on matching across query interfaces.

Other methods apply schema matching network techniques in similar problems: Nguyen et al. [19] face the problem of integrating several product descriptions from online stores into their catalog, which plays the role of a global schema. The work by Toan et al. [20] addresses the problem by using a probabilistic model to find common concepts in different schemas.

Alani and Saad [35] propose a schema matching method to create clusters of elements from multiple schemas, which represent different concepts among them. The method consists of two phases: creating an ontology based on schemas and cluster elements based on lexical and semantic similarities. In the first phase, the method performs a preprocessing of elements by removing stopwords, punctuation, and unwanted characters. Also, it decides which elements shall take part in the clustering stage based on TF-IDF scores. The second phase relies on XBenchMatch [36], a benchmark involving a set of criteria for testing and evaluating schema matching tools. Specifically, the method uses XBenchMatch as a semantic dictionary to help generate semantic similarities. Elements selected in the first phase with high semantic similarity are clustered together. When new elements are matched against the ontology, the method uses Cosine, Dice, and Jaccard to compute lexical similarities. The authors performed experiments in two datasets with F-measure ranging from 0.86 to 0.92. The authors do not clarify if the method generates a final schema. They also do not present statistics about unmatched elements.

The methods above address the schema matching network problem by focusing on creating clusters of similar elements from several schemas. Each one has its particular strategy, such as using statistical information or semantic dictionaries. Some methods make a slight change in the problem and consider that the final global schema is known, matching all schemas against the global schemas. Unlike the clustering methods, our work considers network constraints that are not captured in clusters. For instance, elements *fname* and *lname* from the same schema might end up in the same cluster. Also, our work does not depend on specific domain dictionaries, and it does not consider that a global schema is known. Even though these assumptions can be valid for many domains, such as product information, that do not change over time, we believe that is not the case to consider as applications evolve and became more complex. Besides that, our method considers using other methods to generate base matches and, from that, finds other unknown

matchings. Finally, our method leverages the user experience to help improve matching results without demanding too much user effort compared to previous approaches.

Schema reconciliation networks

The methods proposed over the years can achieve a good performance on some datasets. However, they cannot be expected to yield a satisfactory result in general. Since matchers rely on automatic techniques, their result is inherently uncertain. In practice, data integration processes often include a post-matching phase, in which matchings are reviewed and validated by an expert [9].

In our work, we consider the *schema reconciliation* approach, which adopts a post-matching phase where a human expert reviews, validates, and corrects the generated matchings [37]. The reconciliation phase occurs after a schema matching method is executed as it generates many mismatches and many uncertain matching answers. The generated answers are submitted to a user (or a crowd of users) to an assertion. As evidenced by Duchateau et al. [38], it is most advantageous to assert generated correspondences than providing matching examples, i.e., it is easier to validate or not a discovered mapping than manually browsing large schemas for adding new matches.

As representatives of methods that adopt the reconciliation approach, we highlight the works by Hung et al. [9] and ArgSM [37]. The work by Hung et al. [9] proposes a model for probabilistic matching networks. It takes the output of a matching system, and it constructs a network that contains the schema elements matched and a confidence score associated, both provided by the matching system. The user is requested to assert inconsistencies with the lowest confidence scores; with each new user assertion, scores are recalculated, such as inconsistencies. The method runs until all inconsistencies are solved, or it reaches a user budget.

ArgSM [37] is a framework to help the process of reconciling a network of schemas. Authors propose a representation that captures the expert's beliefs and enables reasoning about their inputs. The framework can detect conflicts in assertions and guide the resolution by re-asking users. Unlike the previous work, ArgSM can leverage a crowd of user experts and manage their opinions. Others consider using a crowd of users when reconciliating a matching network [33, 37]. They consider, besides the assertions of matching answers, the quality of the users based on their domain knowledge and previous assertions.

To address the participation of the specialist in the reconciliation process experiments, we assume as well as all mentioned works that she shall always give correct assertions about matching candidates and we shall consider an unlimited assertion budget. We also address the reconciliation as a step of the matching process. We consider the reconciliation task with a single user with no optimizations regarding the order of questions asked, i.e., we do not intend to minimize the number of asked questions as other works may focus. The reconciliation is only performed to show that our method can achieve even higher results regarding the matching quality. We leave the optimization as future work.

To summarize, the schema matching network problem brings some challenges such as the high number of candidates to evaluate, the network constraints to comply, and the difficulty of having a large number of labeled examples and unbalanced datasets (in the case of machine learning approaches). Our study aims to tackle such issues by taking answers from black-box systems as training examples, using network constraints to help

build a good and informative training set, and finally, still take advantage of the user effort to help improve final matching quality, without making it a laborious task.

Using machine learning in schema matching networks

To address the schema matching network problem, we exploit machine learning techniques to overcome difficulties that arise in this scenario. In this section, we present a family of methods that leverage such an approach. First, we present RF4SM—*Random Forest for Schema Matching*, a machine learning-based technique to address the schema matching network problem. Next, we present RF4SM-B—*Random Forest for Schema Matching Boosting*, which is an extension of the first method that acquires training instances with no labeling effort by taking answers from unsupervised heuristic methods while maintaining matching quality. Finally, we present RF4SM-B-Rec, which brings the specialist back into the process by asking them to assert matching answers produced by our method. By doing so, users can contribute to improving matching quality, but the method spares them from exhaustive work.

Similar to the methods discussed in the "Background and related work" section, we address the schema matching task as a *classification* task, where, given a pair of elements from two schemas $\langle e_{S_1}, e_{S_2} \rangle$ (or a *matching candidate*), the method should assign a label `TRUE` if the elements are a real matching, or `FALSE` otherwise.

We started developing our first method by testing several supervised learning approaches to verify if there is a suitable technique for the schema matching network problem. Considering that the same labeled set of instances is given as the training set of examples, we can generate several models for the matching task and use a ground truth to evaluate such models.

This draft procedure is described in Algorithm 1. For a given matching task, take a random subset containing p% of the candidate pairs of schema elements, annotate these pairs, and generate a labeled set of examples I_{tr} (line 3). Then, carry out a training process according to a supervised learning algorithm and create a model *Sup_Model* (line 5). *Sup_Model* can be used to classify the remainder of unlabeled instances, the test set I_{ts} (line 6).

Algorithm 1 Draft_Algorithm(C, p)

1: Let C be the set of all matching candidates from the matching network
2: Let p is the percentage of training instances used
3: $I_{tr} \leftarrow randomSubset(C, p)$ ▷ random training instances
4: $I_{ts} \leftarrow I \setminus I_{tr}$ ▷ test set
5: $Sup\_model \leftarrow SupervisedLearningAlgorithm(I_{tr})$
▷ Generates models using one of the machine learning methods available
6: $(M_+, M_-) \leftarrow classify(Sup\_model, I_{ts})$ ▷ classified instances
7: **return** M_+ ▷ matching candidates classified as `TRUE`

The "Experimental evaluation" section details an experiment we carried out using this procedure and the results obtained for several well-known learning strategies tested. The best supervised learning algorithm, according to this experiment, was chosen to be the base classifier of the method.

RF4SM—Random Forest for Schema Matching

The experiments for choosing the most suitable approach to the schema matching network problem are further detailed in the "Experimental evaluation" section. We concluded that *Random Forest* was the most suitable algorithm and it was chosen as the base classifier for `RF4SM`. The Random Forest algorithm is simple and flexible enough to be applied to different input types [39] and easy to implement. In addition, other methods, such as Yet Another Matcher [14, 31], the work by Rong et al. [40], ALMa [29], and the work by Reis et al. [41], used forests of trees or random forests as learning algorithms to address the classic schema matching problem. This fact also backed up our decision to go with decision trees as the base learning approach.

Once the base classifier is selected, `RF4SM` can be defined according to Algorithm 2. `RF4SM` receives as input a network of schemas to be matched. By combining all possible pairs of elements between schemas, the algorithm creates a set C of matching candidates. A random subset of the matching candidates is labeled and used as the training set I_{tr}. Next, the `RF4SM` model is generated by training according to the Random Forest technique. Finally, the model is used to evaluate the matching candidates. Those who get the label `TRUE` will compose the answer of the method, i.e., the schema matching network M_+.

Algorithm 2 RF4SM_Algorithm(C)

1: Let C be the set of all matching candidates from the matching network
2: Let p be the percentage of training instances used
3: $I_{tr} \leftarrow randomSubset(C, p)$ ▷ random training instances
4: $I_{ts} \leftarrow I \setminus I_{tr}$ ▷ test set
5: $RF4SM\_model \leftarrow randomForestSupervisedLearning(I_{tr})$
6: $(M_+, M_-) \leftarrow classify(RF4SM\_model, I_{ts})$ ▷ classified instances
7: **return** M_+ ▷ matching candidates classified as `TRUE`

While this method can generate a different model for each new matching task, it may be problematic in some tasks to learn good models with a small number of examples. `RF4SM` can be a suitable option when a crowd of users is available to provide large amounts of matching examples ([33, 37]). However, that might not always be the case. To cope with that scenario, we evolved our method to take into consideration two aspects: respecting network constraints and obtaining a large and reliable amount of training examples.

RF4SM-Boosting

To obtain training examples without user labeling, `RF4SM-B` (RF4SM-Boosting) uses heuristic methods to generate such examples. As these heuristics usually generate unreliable matchings (false positives), `RF4SM-B` cleans the set of generated examples using the network constraints presented in the "Schema matching networks" section as filters.

The rationale is to avoid false positives in the training set and possibly to learn better models. Also, network constraints help identify negative examples. Thus, this strategy is useful for generating both positive and negative training examples. For example, in Fig. 3, if the matching a is given as a `TRUE` matching, we can automatically label matching

candidates b and c as `FALSE` matchings. We obtained three labeled instances with only one given label.

We describe RF4SM-B in Algorithm 3. First, the algorithm submits the input set of schemas to a heuristic base method. Such a method is treated as a black box to our method. This method generates a set of possible positive matches A_+ that is filtered using the network constraints. The filtering generates a set of accepted matchings A_+, a set of rejected matchings A_-, and a set of uncertain matchings U. The accepted matchings are the ones that do not conflict with any other. The rejected matchings are automatically generated using the network constraints as filters, as described in the example of Figs. 3 and 4. The uncertain matchings are the set of matchings that conflict with each other. The accepted and the rejected matchings are used in the training step. The `RF4SM-B` model is then generated using the Random Forest algorithm. Finally, the model is used to evaluate the matching candidates. Candidates classified as `TRUE` matchings compose the schema matching network.

Algorithm 3 RF4SM-B_Algorithm(C, A_+)

1: Let C be the set of all matching candidates from the network
2: Let A_+ be the set of positive answers given by the base method
3: $(A_+, A_-, U) \leftarrow networkConstraintsFilter(A_+)$
▷ Network constraints produce three sets: A_+, A_- and U
▷ A_+: matchings labeled `TRUE` and accepted by the network constraints
▷ A_-: matchings rejected by the network constraints.
▷ U: uncertain matchings, i.e., they conflict with each other.
4: $I_{tr} \leftarrow A_+ \cup A_-$ ▷ the training set is the union of positive and negative sets
5: $RF\_model \leftarrow randomForestSupervisedLearning(I_{tr})$ ▷ Training
6: $(M_+, M_-) \leftarrow classify(RF\_model, I)$ ▷ Classifying instances
7: **return** M_+ ▷ Matching candidates classified as `TRUE`

The `RF4SM-B` method is an evolution of `RF4SM` that eliminates the user effort in labeling examples by using the answers of a black-box system. A further improvement would be to ask the user to validate the results in a task called *reconciliation*. This task is usually less laborious than labeling training examples.

RF4SM-B-Reconciliation

As it occurs after the schema matching process finishes, the reconciliation task can be optional. However, it is a useful step as it can help to improve the results of a matching network process. The user participates in the process by reconciliating matching answers, i.e., the user acts by accepting or rejecting matching answers (or *matching candidates*).

This process does not require much knowledge from the user (e.g., compared to tuning the parameters of a method) apart from knowing the schemas involved. Also, it is shorter than the job of labeling all training examples. In fact, the number of training instances is often even higher in the networked scenario, making the labeling process more prone to human errors.

In `RF4SM-B-Rec`, the user is invoked only to assert uncertain matchings, i.e., the ones that conflict with themselves. Hence, their job is less laborious than classic approaches

when they would review all generated matches. In Algorithm 4, we describe this approach. After running `RF4SM-B`, the set of uncertain matchings U is submitted to user assertion, and the algorithm can incorporate accepted answers by the user (U_+) in the final matching network (M_+).

Algorithm 4 RF4SM-B-Rec_Algorithm(C, A_+)

C is the set of all matching candidates from the network
A_+ is the set of positive answers given by the base method
1: $M_+ \leftarrow$ *RF4SM-B_Algorithm*(C, A_+) ▷ presented in Algorithm 3
2: $(M_+, M_-, U) \leftarrow$ *networkConstraintsFilter*(M_+)
▷ The network constraints produce three sets of instances :
▷ M_+ (accepted), M_- (rejected) and U (uncertain)
3: $(U_+, U_-) \leftarrow$ *userReconciliation*(U)
▷ The user accepts or rejects instances in the reconciliation process
4: $M_+ \leftarrow M_+ \cup U_+$
5: **return** M_+ ▷ matching candidates classified as `TRUE`

In the "Evaluating RF4SM-B-Rec" section, we present experiments to verify the improvement provided by a user when reconciliating the results of `RF4SM-B`. We compare the effort made by a user when reconciliating automatic methods and `RF4SM-B` and how the use of different base systems can impact the final stage.

Experimental evaluation

Aiming at validating our assumptions and claims, we performed an extensive set of experiments to evaluate our methods when running over datasets previously used in schema matching experiments reported in the literature.

Datasets

All of the experiments we carried out used a collection of five datasets. Each dataset represents a schema matching network task and contains more than two schemas. All of the schemas in a dataset belong to the same domain. Table 2 presents an overview of the characteristics of the datasets.

These datasets were used previous in experiments reported by [42] and [11] (betting); [43] and [9] (business); [18] (magazine, book); [16] (book); and [9],[2] and [29] (order).

We stress that, in all cases, the actual number of matches to be found is very low compared to the number of candidate matches, making each task a challenging one. Even

Table 2 Experimental datasets and their characteristics, organized by domain

Domain	Betting	Business	Magazine	Book	Order
#Schemas in the network	12	3	12	4	5
#Matching tasks	66	3	66	6	10
#Candidate matchings	45013	20840	87200	7882	49192
#Correct matchings	863	177	810	44	298
#Elements per schema (avg)	26	84	36	36	72
#Elements with a match (avg)	18	74	16	9	46

though these datasets have been used in previous work, some were made available without their ground truths. Therefore, we had manually matched the schema networks before performing our own experiments. Also, the ground truths of every matching network are guaranteed to comply with all the constraints in the network.

Evaluation metrics

Let M be the set of all matching candidates, and $C \subset M$ be the set of correct matchings for a given matching network. Let A be the set of all answers given by a method, i.e., the set of matches generated by some method. We evaluate the effectiveness of a method according to *precision* $\left(\frac{|A \cap C|}{|A|}\right)$, *recall* $\left(\frac{|A \cap C|}{|C|}\right)$, and *F1-score* $\left(\frac{2 * precision * recall}{precision + recall}\right)$. Notice that, due to unbalancing number of valid and invalid matching candidates, the invalid candidates being the vast majority, we only consider the positive candidates in our evaluation. Furthermore, finding valid matching candidates is the real goal of a matching task.

Precision and recall are largely used in the schema matching field. However, they cannot be used alone to evaluate the quality of a matching network. Precision measures how correct the correspondences returned are, i.e., the higher is the number of correct correspondences returned by a method, the higher is its precision value. Recall measures how much the set of answers returned to cover the perfect solution. In other words, the lower the recall, the higher the number of correct correspondences missing.

The F1-score is often used to compact both metrics in one single value. It is a harmonic mean of the two values. It treats both precision and recall as equally important scores. Duchateau et al. [38] argue that recall is more important than precision from the user point-of-view, as it is harder to find undiscovered correspondences than to assert incorrect correspondences returned by a method. In this work, we do not intend to propose a new evaluation method. Hence, we rely on F1-score to represent an overall quality measure of matching results.

Base systems

As described in the "Using machine learning in schema matching networks" section, our methods `RF4SM-B` and `RF4SM-B-Rec` use a "black box" system as its base provider of matching candidates. As these base systems, we used unsupervised methods that use heuristic approaches, namely, the well-known methods COMA/COMA++ [2, 11] and Similarity Flooding (SF) [7]. Both base systems are available online. [3][4]

We selected the library of matchers available in COMA/COMA++ and Similarity Flooding to be the base matchers for all learning techniques we experimented with. All of the answers given by the methods were collected, and those were used to build a network of answers which can be evaluated according to the metrics mentioned above.

In COMA/COMA++, we used the best combination of matchers and parameters reported by the authors [2, 11]. They report that in its best configuration, COMA runs all the hybrid matchers, aggregates matrices by their Average, matches in *both* directions, and selects matches using the *MaxDelta* strategy ($max = 1$ and $\Delta = 0.02$).

All of the results reported for Similarity Flooding were obtained by running the method in its default configuration. Similarity Flooding runs by making a *string matching*

[3] https://sourceforge.net/p/coma-ce
[4] http://infolab.stanford.edu/~melnik/mm/rondo/

between schema element names, then applies the *flooding algorithm* that propagates the similarities through nodes in the graph. Finally, it applies filters to select the matchings.

We note the base systems are treated here as black boxes, i.e., we can change them for any system that generates matching candidates. For this study, we chose to use two of the best-known unsupervised matching systems. As they are unsupervised, the user input does not impact the evaluation regarding the number of user inputs. All of the results reported for machine learning algorithms are the average of 30 runs, each one with a different random seed.

Baselines

As representatives of heuristic approaches, we also use COMA and Similarity Flooding (SF) as baselines. To the best of our knowledge, both methods were never presented addressing the schema matching network problem. However, they can be adapted to run in the networked setting by running each method taking every combination of pairs of schemas (classic schema matching) and creating a network of matching answers.

There are several machine learning methods in the literature for the classic schema matching scenario [14, 19, 28, 29]. Thus, instead of adapting each method to the networked scenario, we constructed network-oriented methods with their core classification algorithms and ran experiments with them.

The methods for the classic schema matching use many different kinds of machine learning algorithms: the method by Gal [28] uses the statistical-based model *Naive-Bayes*; YAM [14] uses tree-based algorithms such as *J48* and functions such as *Logistic Regression* and *Bayes Networks*; the method by Nguyen et al. [19] also uses *Regression* and ALMa [29] is also based on a tree-like model.

Evaluating machine learning algorithms for the schema matching networks

In this section, we evaluate different machine learning methods previously used for the classic schema matching scenario in the context of the schema matching network task. This experiment resulted in choosing *Random Forests* as our base classifier. This experiment also provides our machine learning baseline. The second experiment tested the random forests against the base methods (`COMA` and `SF`) to verify if the learning method can achieve results comparable to the heuristics used by the family of `RF4SM` methods.

Setup. We performed experiments with several machine learning algorithms featured in the Weka package [44]. We ran each classifier with five different sizes of training sets, ranging from 10 to 50%. We note that 50% of labeled matching instances can represent more than 40,000 candidate pairs for the schema matching datasets we use, which is a high number of labeled instances. Realistically, this number of labeled examples seems unfeasible once that examples from one matching task can hardly be used in another task of a different domain. The remainder of the instances were used in the test set. All the results reported are an average of 30 runs. We tested all the algorithms in our five experimental datasets. For the first experiment, the average of the results in the five datasets corresponds to the final result. The algorithms have been used in learning approaches to the matching problem. For details of the parameters required by algorithms, please refer to the Weka documentation.[5] The specific settings we used for each algorithm are as follows:

[5] http://weka.sourceforge.net/doc.stable/

- *AdaBoost: weightThreshold*=100, *baseClassifier*= DecisionStump, *numberOfIterations*=10;
- *J48: unpruned=false*, *confidenceFactor*=.25, *subtreeRaising=true*, *binarySplits=false*;
- *Logistic regression: maxIts*= − 1, *ridge*=1.0*E* − 8;
- *Random forest: numTrees*=100, *maxDepth=unlimited*;
- *Random tree: KValue=random*, *maxDepth=unlimited*, *minNum*=1.0

Validating RF4SM

In this first experiment, we wanted to test which of a variety of popular classifiers were best suited to the task of matching networks of schemas. We randomly divided the instances into two groups for each of the five datasets: training and test sets. The training set is given to each classifier to build models, then the test set is submitted to the models, and we evaluated the classification according to our metrics. We repeated the experiment with 30 different random seeds, varying the size of the training sets.

We report the average F1-score obtained by the classifiers in Fig. 5. The Random Forest classifier achieved the highest average of F1-score, consistently improving when increasing the training set. With this experiment, we established the Random Forest model as the base classifier for the method. We point out that works such as YAM [14] and ALMa [29], which address the classic schema matching task, also chose ensembles of trees as their base classifier. From this point on, we referenced our method as Random Forest for Schema Matching—`RF4SM`.

The learning strategies tested in this experiment are featured in several works that address the classic schema matching task [14, 19, 28]. With that in mind, we decided to take the learning algorithms as the representatives of these methods and considering them as the baselines for the schema matching network task.

RF4SM and heuristic strategies—base methods

The next experiment was designed to evaluate the base classifier against the base methods. We ran `RF4SM` with training sets of different sizes ranging from 10 to 50% of each

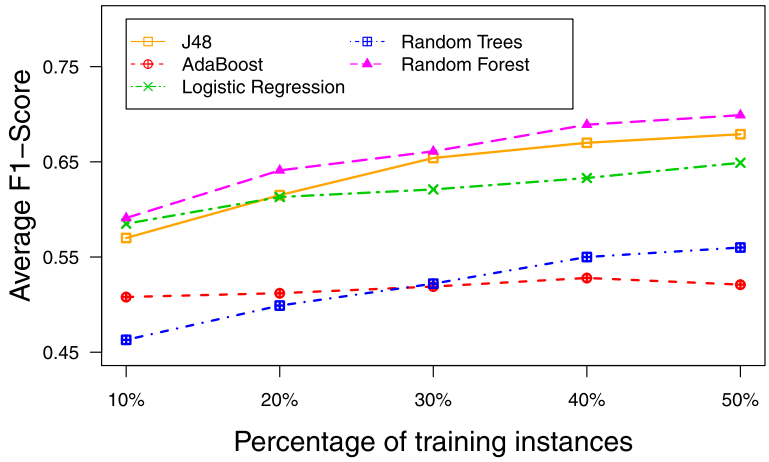


Fig. 5 Avg. F1-scores by each classifier for training sets of different sizes

experimental dataset. This experiment aimed to estimate the volume of learning instances required to achieve results comparable to the base methods. This volume ultimately indicates the effort needed by a user for labeling examples before the matching process in a supervised setting.

In absolute numbers, labeling 10% of instances means a user has to label at least 700 instances (in the smaller dataset), which is a high number of instances to label. We considered 50% was a big enough number of labeled instances to have available, even in a hypothetical situation. As we already stated, having such a cumbersome task can lead to human errors. Therefore, our study aims to ease the user effort as much as possible while reaching high matching quality levels.

For brevity, we opted to show only the F1-score values achieved by `RF4SM` using 30%, 40%, and 50% of training, alongside the base methods in Fig. 6. On average, `RF4SM` with 50% of training instances reaches F1-score of 0.70, topping `COMA`'s F1-score (0.68), `SF` reached 0.65 trailing `RF4SM`-30% (0.66). We also point out that `RF4SM-40%` achieved the highest average precision (0.78), and `RF4SM-50%` achieved the highest precision in a single task (0.89).

In absolute numbers, `RF4SM` needed an average of 16800 examples to train its models, which is a high number of instances to get with a high cost as they must be labeled. The next set of experiments addresses this issue.

RF4SM-B: experimental evaluation

We designed the following experiments to test our boosting strategy. We performed an experiment to verify differences in the training sets, such as accuracy and quantity of positive examples. Additionally, we evaluated how `RF4SM-B` compares to previous methods.

Setup. In this set of experiments, we evaluated the methods in all five datasets (presented in Table 2). As `RF4SM-B` contains a random component, all of the results of the method are the average of 30 runs. We ran `RF4SM-B` using the two different base systems,

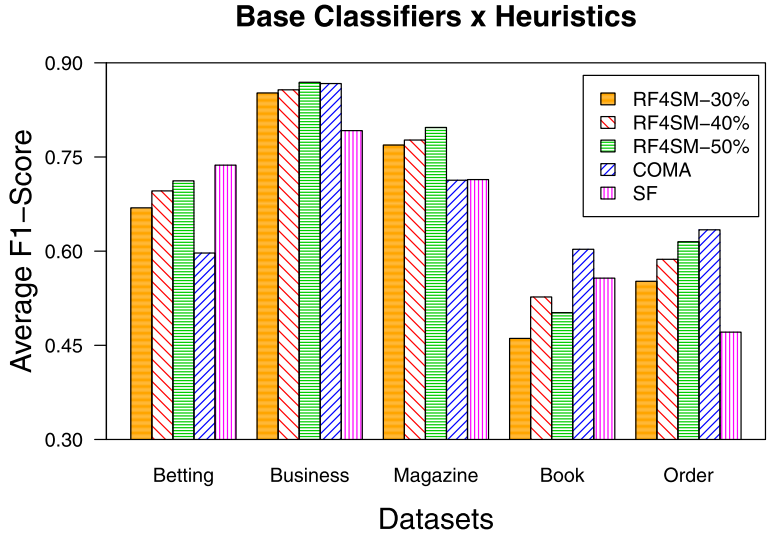


Fig. 6 Average results of supervised and heuristic methods in the five datasets

COMA and SF and trained the models the training sets generated by each of these methods. Notice that we used COMA and SF both as base methods and as baseline heuristic methods for this set of experiments.

Obtaining training instances

The network constraints presented in the "Schema matching networks" section are good indicators of the integrity of the matching network. When the matching network has a high number of inconsistencies, i.e., the network is uncertain, it is an indication that the network has many mismatches.

In this experiment, we want to inspect the networks of matchings created by the answers of the base methods to assess if they have a notable number of inconsistencies and how large this number is. Ultimately, this network of answers will be used as the automatically labeled training set for RF4SM-B.

To build the networks of answers for each of the five domains, we submitted every combination of pairs of schemas to the heuristic methods. After that, all of the methods' answers were gathered and inserted into a network of answers. We counted every time a match candidate m violated a constraint. We also had the counters divided by the type of constraint violated. Table 3 presents all the results. Type I constraints refer to *one-to-one* constraints, and type II constraints refer to *cycle* constraints.

Training with filtered instances

To be able to provide the automatically labeled examples to RF4SM-B correctly, we used a filter based on the network constraints to prune inconsistent matchings of the training set. In the next experiment, we compare how RF4SM-B performs using *filtered* and *unfiltered* training sets against baselines.

We report two runs of RF4SM-B, each one with a different base system. We compared the results against the base system (heuristic baseline) and the RF4SM (supervised learning baseline). The first run uses COMA as a base system, and the F1-score values achieved are depicted in Fig. 7. The second run uses Similarity Flooding as a base system, and the results are reported in Fig. 8. In both cases, we verified the statistical significance of the results using the Welch two-sample t-test, with all p-values well below 0.05.

RF4SM-B can achieve better results of the F1-score in the majority of the datasets. On average, it outperforms both base systems, COMA and SF, and RF4SM while using no user input. Still, regarding the training set, the filtering process managed to acquire around 40% of labeled examples on average (similar to RF4SM). Yet, those labels are not validated by a user and are not guaranteed to be 100% correctly labeled.

We also notice that using the *filtered* training set helped to learn better models as evidenced by higher precision values without compromising recall. When the number of inconsistencies increases, so does the gain observed in the matching results.

Table 3 Number of constraints violated per matching network task

	Dataset	Betting	Business	Magazine	Book	Order
COMA	Type I	5	2	6	6	15
	Type II	248	20	278	16	211
	Total	253	22	284	22	226
SF	Type I	0	0	0	0	0
	Type II	364	0	305	10	72
	Total	364	0	305	10	72

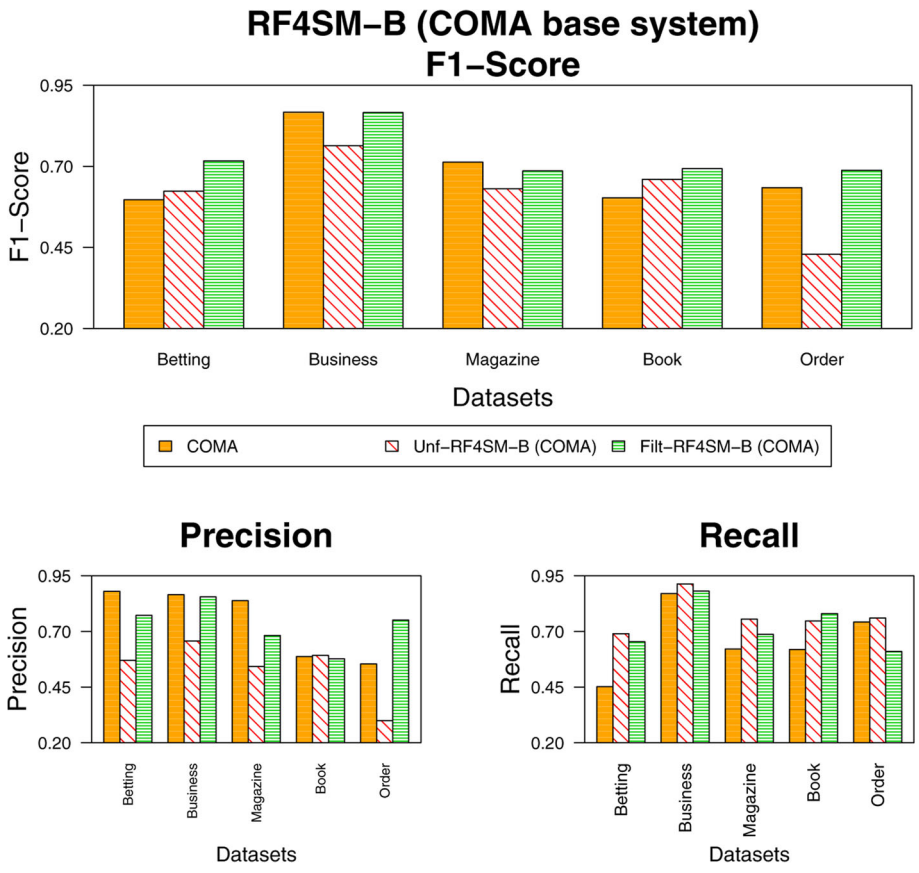


Fig. 7 `RF4SM-B` (*unfiltered* and *filtered*) compared to `RF4SM`, with `COMA` as the base system

`RF4SM-B` managed to train a random forest without asking the user for labels. However, users can be a valuable source of information, and they can still enhance matching results. Hence, in the next step, we will include user knowledge in the process by asking them to validate matchings in the reconciliation process.

Evaluating RF4SM-B-Rec

After successfully acquiring training instances sparing users of the job of labeling them, we wanted to improve the matching results by using their powerful source of information by asking to reconciliate the network of schemas generated. The next experiment was designed to verify if `RF4SM-B-Rec` can achieve better results than reconciliating from the base systems.

For this experiment, we took the network of answers from our method, separated the uncertain candidates (the ones that have a conflict with another answer), and submitted them to user assertion. The user can validate a candidate as a correct answer or reject it and remove it from the pool of answers. After the reconciliation, we measured precision, recall, and F1-score.

Setup. In this set of experiments, we evaluated the methods in all five datasets from Table 2. As `RF4SM-B-Rec` contains a random component, all of the results presented are the average of 30 runs. The settings of other methods featured in this section remains the

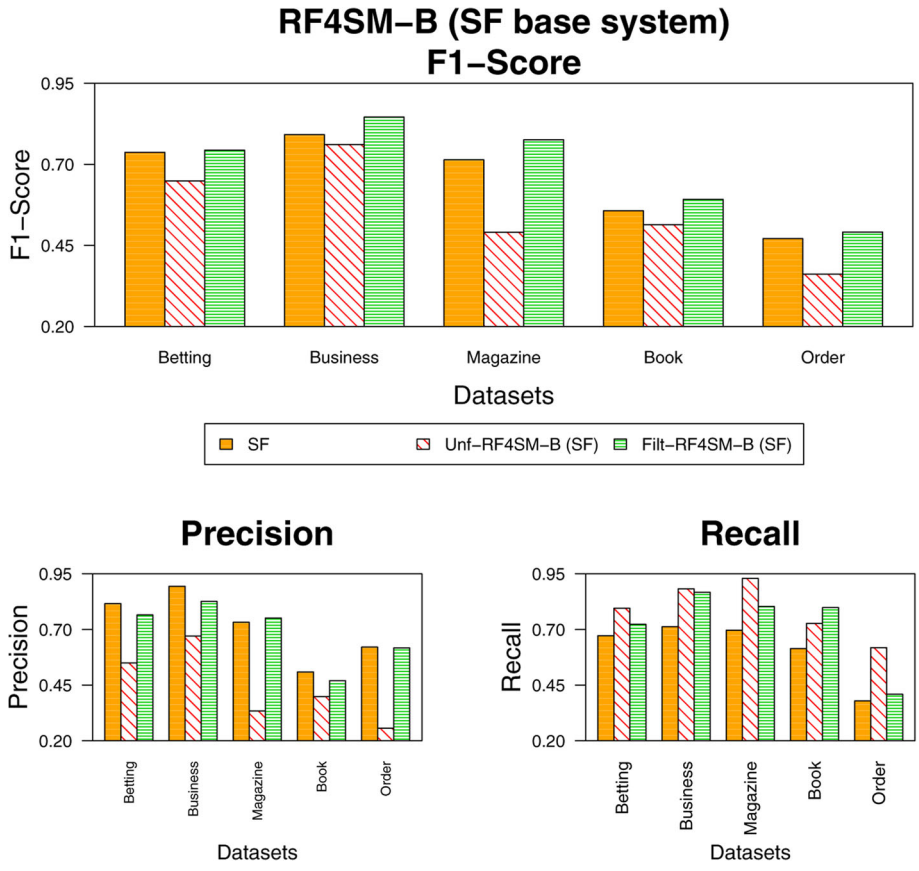


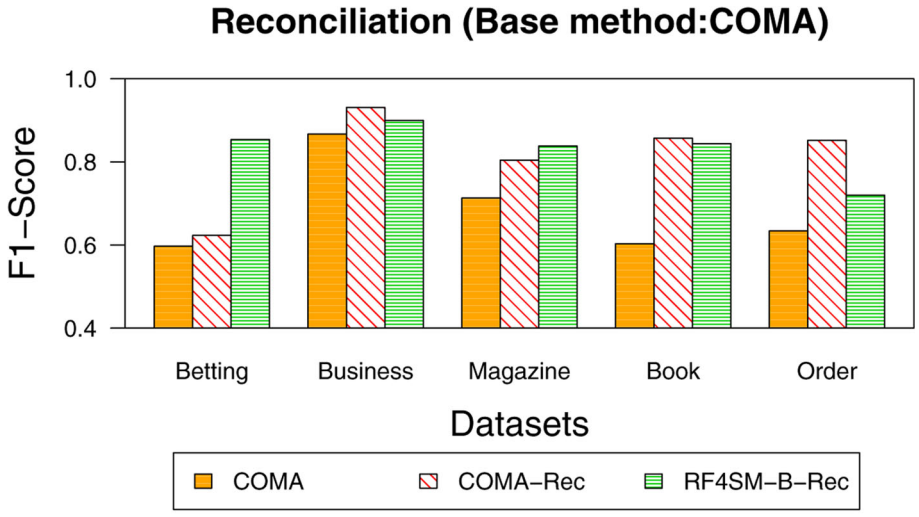
Fig. 8 `RF4SM-B` (*unfiltered* and *filtered*) compared to `RF4SM`, with `SF` as the base system

same as in previous experiments. In this study, we consider that the users give only reliable assertions, i.e., if they confirm a candidate as a `TRUE` matching, then we can assume that the matching is `TRUE`. The same applies to `FALSE` matchings. As we did not use any optimization strategy aiming at minimizing user effort, the order in which matching candidates are presented for review does not affect the outcome.

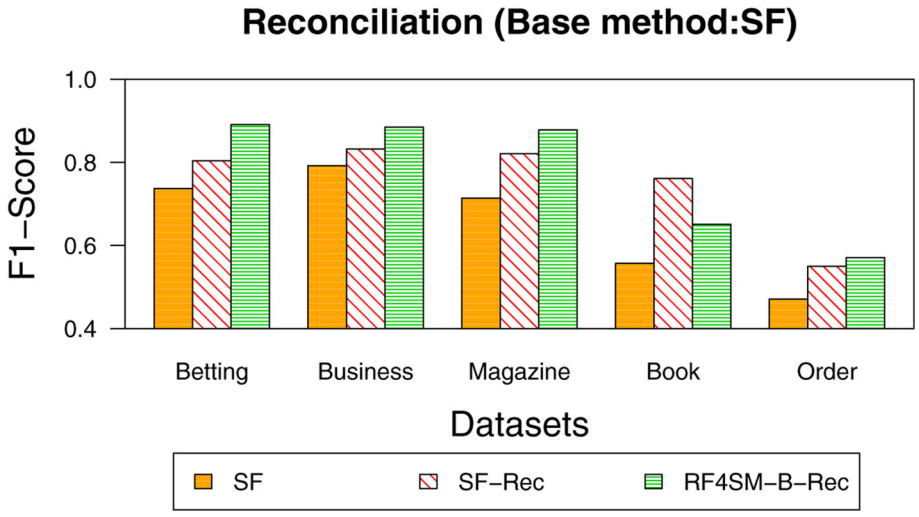
We present the results of this experiments in Fig. 9a and b, for which we also verified statistical significance using the Welch two-sample t-test, with all p-values well below 0.05.

We show the F1-scores reached in Fig. 9a when we reconciliate the network of answers from `RF4SM-B` trained with data from `COMA`. On average, the RF4SM-B reconciliation reaches an F1-score of 0.83 against a score of 0.81 from `COMA`'s reconciliation. However, `COMA`'s reconciliation requires much more user intervention, as shown in Fig. 10a. Besides similar performance considering the F1-scores, from the point of view of an expert that performs the reconciliation manually, it is most advantageous to use `RF4SM-B`, as its task is reduced by five times, i.e., they have to label five times more matching candidates manually.

We also performed a similar experiment considering the automatic learning set gathered from Similarity Flooding's answers. The F1-scores reached are shown in Fig. 9b. On average, `RF4SM-B` achieved an F1-score of 0.77 against 0.75 from Similarity Flooding. The behavior was similar to that observed in COMA's experiment. The user had to label



(a) COMA as the base method.



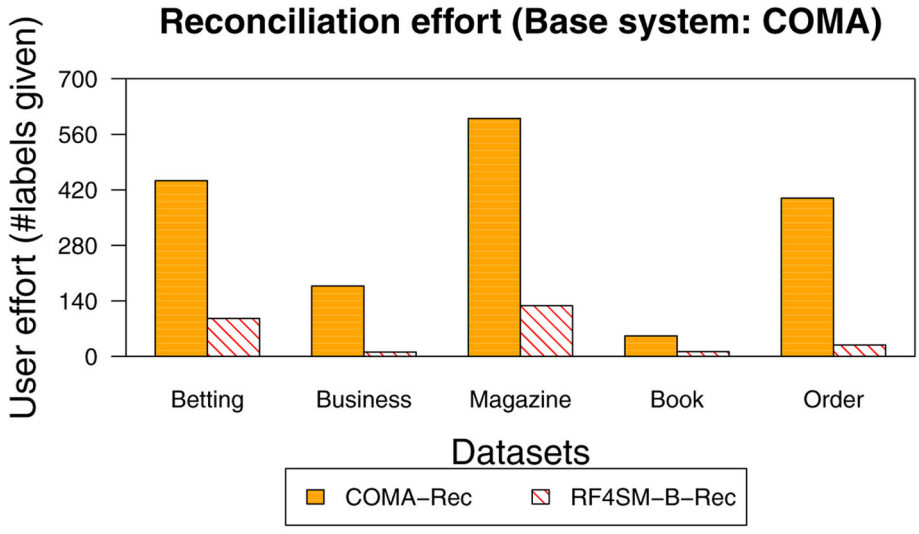
(b) Similarity Flooding as the base method.

Fig. 9 F1-scores achieved when performing reconciliation at the end of the process

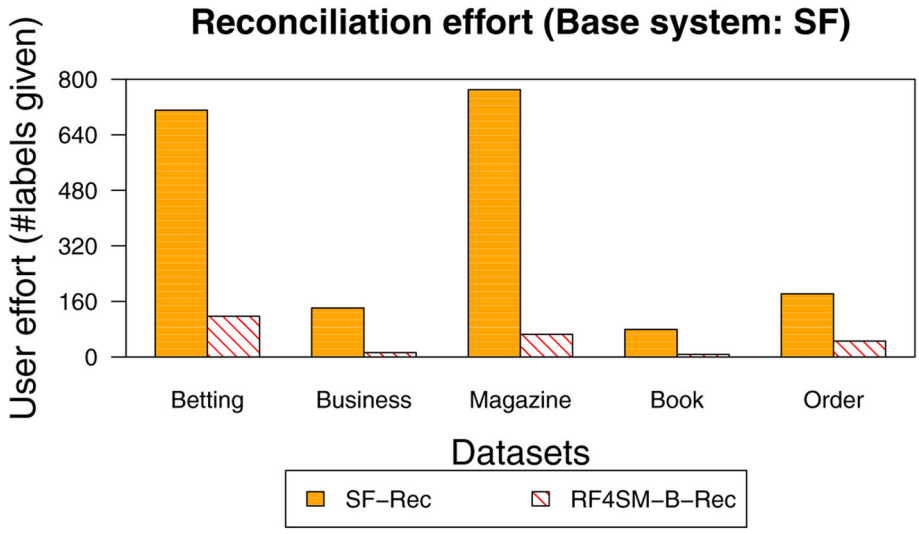
a larger amount of Similarity Flooding's answers to reach a comparable F1-score. The amount of labels given by the user in the reconciliation task is given in Fig. 10b. In this experiment, the user effort also was reduced by approximately six times.

For both base systems, `RFMS-B-Rec` managed to find more matchings that were undiscovered previously, hence achieving higher values of recall, with an average of 0.77 against 0.69 (`COMA-Rec`) and 0.61 (`SF-Rec`). Also, `RFMS-B-Rec` maintained a high average precision (0.89), compared to the perfect precision, as if the reconciliation was performed with the base method answers.

Finally, we compare the results of `RF4SM-B` and `RF4SM-B-Rec` to see how much the reconciliation process can improve the result of our method. We show the results of



(a) Reconciliating networks generated using COMA.



(b) Reconciliating networks generated using Similarity Flooding.

Fig. 10 User effort in the reconciliation of the networks

precision, recall, and F1-score for the methods using `COMA` as the base system in Fig. 11, As expected, the reconciliation process leads to better results, achieving an average F1-score of 0.83 while asking for a low number of labels from the user, i.e., 55 per dataset on average.

The same behavior can be observed when analyzing the results of the method with `SF` as the base system. Figure 11 presents these results. On average, `RF4SM-B-Rec` achieved an F1-score of 0.77, and the user labeled 49 candidates per dataset, indicating that our method can be effectively applied with different unsupervised schema matching systems as a black box. Also, by comparing the results obtained with both base systems, the better the base system, the better will be the boosting provided by `RF4SM-B-Rec`.

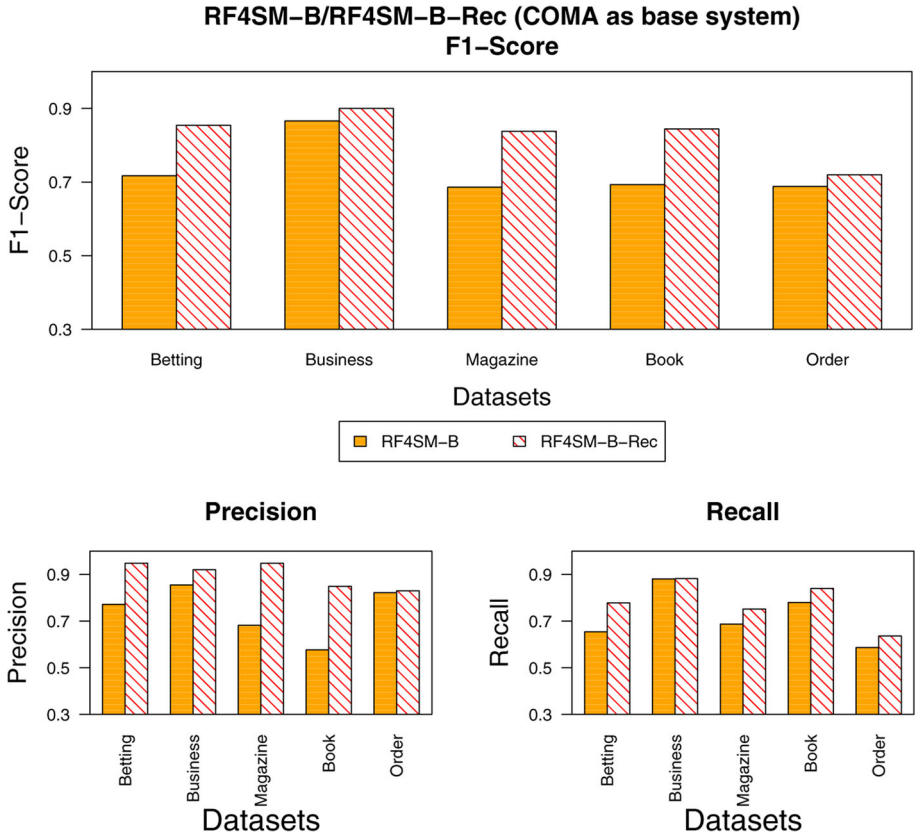


Fig. 11 Comparison between `RF4SM-B` and `RF4SM-B-Rec` using `COMA` as the base system

As before, the results we present in Figs. 11 and 12 have statistical significance verified by the Welch two-sample t-test, with p-values <0.05.

Summing up

To sum up the results from our experiments, Fig. 13 shows the average F1-score of the methods discussed in the sections above and divided them into groups, according to the type of user interaction in the matching task, depending on how much user effort they use: *exhaustive, none, or some* user effort.

The first group, "Exhaustive User Effort," consists of the supervised method `RF4SM` running with different sizes of training. Methods in this group require an expert to provide a training set containing a massive amount of labeled examples. Although `RF4SM` can achieve higher values in the metrics than the other methods, it required at least 10% of datasets, which could mean at least 8000 labeled matching examples. This can be unfeasible for typical requirements of real applications. As we could observe in Fig. 6, the more training, the better the model generated and the better the matching quality regarding F1-score. Generally, the learning approaches achieve better results than heuristics. However, they require that a large amount of training data is available, which might not be true for some matching tasks.

The group "No User Effort" includes the heuristic methods, `COMA` and `SF`, and the semi-supervised boosting-based method `RF4SM-B`. None of the methods uses any input from

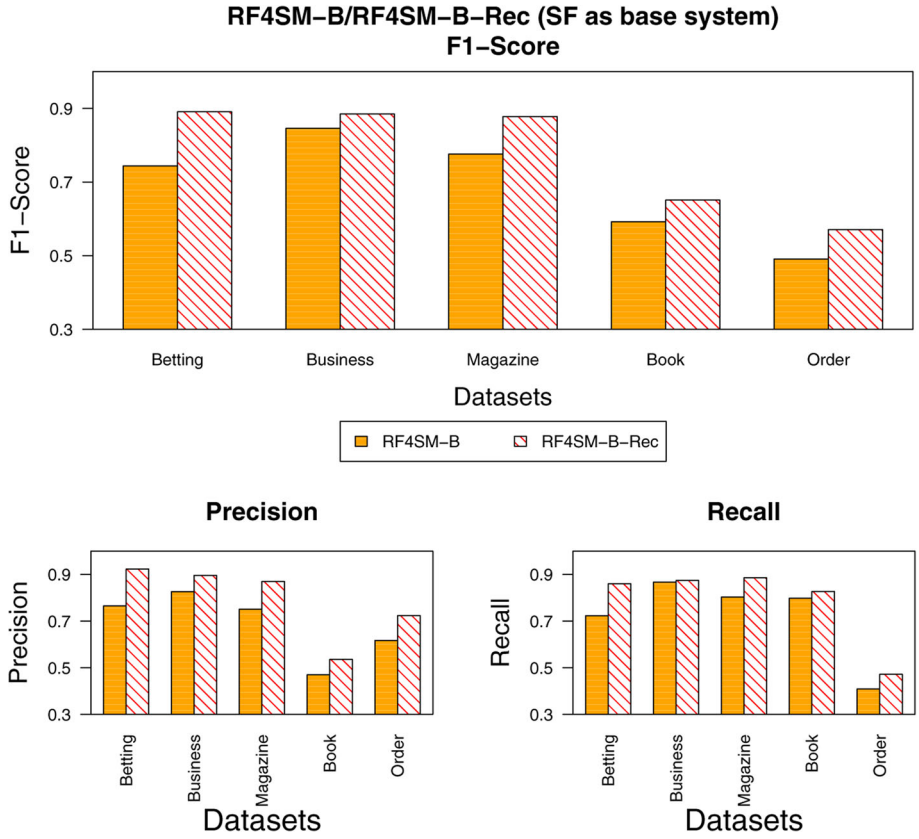


Fig. 12 Comparasison between `RF4SM-B` and `RF4SM-B-Rec` using `SF` as the base system

experts (as in labeling matching examples), even though they can have parameters tuned before their execution. Typically, the answers provided by these methods are reviewed by the expert after the execution. `COMA` and `SF` take into consideration the hierarchical aspect of the schemas, but they do not consider the network constraints. Hence, they rely on their heuristics that prioritize precision, while `RF4SM-B` can create more gen-

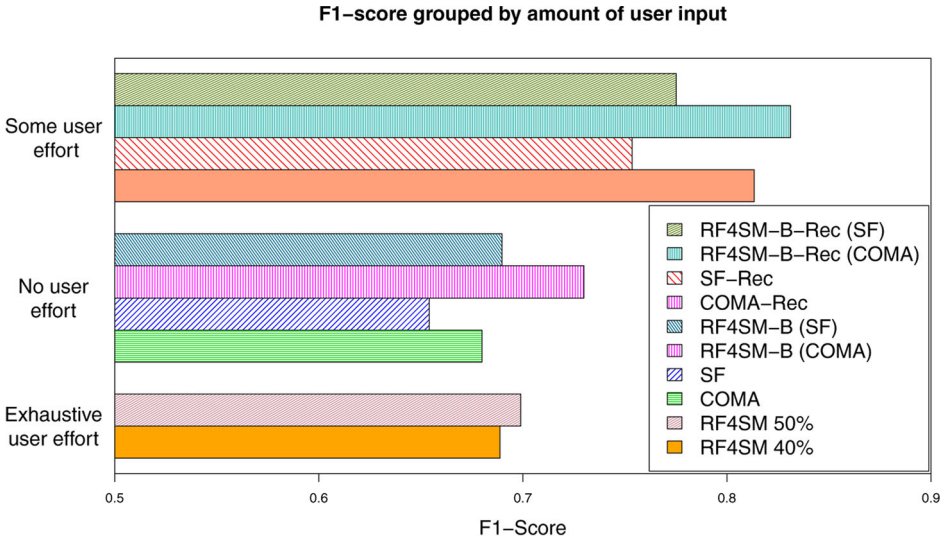


Fig. 13 F1 values by the methods according to the type of user participation in the process

eral models that may lack precision in favor of recall but whose output can be validated later. Usually, the methods in this group can produce good matching results. However, to achieve higher levels of results, they rely on a specific tuning of parameters. The tuning usually requires that a domain specialist is available; it also requires the knowledge of the inner workings of methods. In addition, tuning performed for one matching task may not produce the best results for a different task, making this hard to generalize.

The group "Some User Effort" includes methods that use the reconciliation stage. Both COMA and SF can improve their previous results but at a higher cost of user effort. On average, the user reviews over 300 candidate matches. As RF4SM-B-Rec uses the network constraints to prune their answers, a high number of answers can be automatically invalidated. Besides, the network constraints help to gather a more reliable automatic training set of examples, thus generating better learning models. By invalidating answers, the constraints also help to ease the user effort. RF4SM-B-Rec reduced the reconciliation task by at least four times in each dataset. RF4SM-B-Rec was able to achieve slightly higher values of F1-score while reducing the effort needed to validate the final matching answers. It can potentially be used with any base system as a black box. We also note that the methods in this group produce better matching results compared to the ones in the "Exhaustive User Effort" group, even though they use a smaller amount of user inputs. On average, 300 candidate matchings were reviewed in this group, which is a much smaller number than 10% of the smaller dataset available (Book dataset), which corresponds to around 700 examples.

Notice that although it is not common to group and compare heuristic methods to machine learning methods, we choose to do it intending to see how much different amounts of user effort can interfere in matching results. In addition, we show that more user participation does not translate to better results, and we can achieve better matching results if using the user input in a more intelligent fashion aligned with network constraints.

Conclusions and future work

In this paper, we presented a study with a family of methods that apply machine learning techniques to the schema matching network problem. The methods carry out the matching process in a network of schemas while respecting network integrity constraints. Such constraints define rules to guarantee that the matching network remains consistent. Also, they might be used to help prune erroneous matches that may be generated by schema matching methods such as ours.

First, we presented RF4SM, which uses machine learning algorithms to find correspondences between several schemas that form a network. In our experiments, we showed that the method is suitable for the problem of matching networks. The method reaches precision values around 0.70, the highest average among the systems tested.

Next, we presented RF4SM-B, which is an evolution of the previous method. It addresses the obstacle of not having a large number of labeled training examples by using heuristics methods to generate training examples automatically. We showed how the network constraints can be used to identify training examples that are likely to be incorrect and the impact they may have on the final results. By using the constraints, we could produce more reliable training sets and using a black-box system; we could spare the user

of the job of labeling matching examples. As a result, on average, `RF4SM-B` achieved an F1-score of around 0.73.

Finally, we present `RF4SM-B-Rec`, which addresses the use of reconciliation in the matching process. In this stage, users are invoked to assert the matchings generated by our method, and they can accept or reject these matchings. As previous methods do not consider integrity constraints, they overlook this valuable source of information to prune wrong correspondences. In our experiments, we show that `RF4SM-B` results can be even better with the reconciliation process. We also show that F1-scores achieved are higher than other methods while asking for a lower number of labels. The experiments show that the user effort in the task can be reduced up to 6 times.

As future directions of our work, we recognize that there is still room for improvement. As evidenced by Hung et al. [9], the order by which the user asserts matchings can be optimized if they are guided to solve inconsistencies in the network. In other words, this means their effort can be reduced even more to achieve the same F1-scores.

We also point out that improvements can be made in aspects related to the machine learning techniques we use. In this study, we considered it out of scope to tune such methods or to find more complex methods. Currently, we used classic machine learning techniques that were largely used in previous methods in the classic schema matching field. We recognize that new methods are emerging and they may yield to ways for obtaining even larger amounts of training data, which is one of the main challenges of the schema matching network problem; also, learners that handle unbalanced classes might be considered as feasible options.

Finally, as seen in the experiments, a reliable training set leads to a more consistent network of matchings. Since we use a black-box system to generate the initial training set of examples, we could also study changing the base system to tuned versions and see how the learning is impacted by it.

Acknowledgements
The authors would like to thank the authors of COMA/COMA++ [2, 11] and Similarity Flooding [7] for making their code available online.

Authors' contributions
Both authors developed the whole work, discussed the results, and contributed to the final manuscript. The authors read and approved the final manuscript.

Funding
This work was supported by projects SocSens(CAPES/PCGI 88887.130299/2017-01), MOBILIS (FAPESP MCTIC/CGI,2018/23064-8), MMBIAS (FAPESP MCTIC/CGI, 2020/05173-4), and by authors' individual grants from CNPq and CAPES.

Availability of data and materials
All datasets used in the experiments are available online, as indicated in the "Experimental evaluation" section.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 2 March 2021 Accepted: 25 October 2021
Published online: 23 November 2021

References

1. Bonifati A, Velegrakis Y (2011) Schema matching and mapping: from usage to evaluation. In: Proceedings of the 14th International Conference on Extending Database Technology. Association for Computing Machinery, New York. pp 527–529

2. Do H-H, Rahm E (2002) COMA: a system for flexible combination of schema matching approaches. In: Proceedings of the 28th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco. pp 610–621
3. Madhavan J, Bernstein PA, Rahm E (2001) Generic schema matching with cupid. In: Proceedings of the 27th International Conference on Very Large Data Bases. The VLDB Endowment, New York. pp 49–58
4. Doan A, Domingos P, Halevy AY (2001) Reconciling schemas of disparate data sources: a machine-learning approach. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. Association for Computing Machinery, New York. pp 509–520
5. Bernstein PA, Madhavan J, Rahm E (2011) Generic schema matching, ten years later. PVLDB 4(11):695–701
6. Doan A, Halevy AY, Ives ZG (2012) Principles of Data Integration. Morgan Kaufmann, San Francisco
7. Melnik S, Garcia-Molina H, Rahm E (2002) Similarity flooding: a versatile graph matching algorithm. In: Proceedings of the 18th International Conference on Data Engineering. IEEE Computer Society, New York. pp 117–128
8. Li Y, Liu D-B, Zhang W-M (2005) Schema matching using neural network. In: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence. IEEE Computer Society, New York. pp 743–746
9. Hung NQV, Tam NT, Miklós Z, Aberer K, Gal A, Weidlich M (2014) Pay-as-you-go reconciliation in schema matching networks. In: Proceedings of the IEEE 30th International Conference on Data Engineering. IEEE Computer Society, New York. pp 220–231
10. Popa L, Hernadez MA, Velegrakis Y, Miller RJ, Naumann F, Ho H (2002) Mapping XML and relational schemas with CLIO. In: Proceedings of the 18th IEEE International Conference on Data Engineering. IEEE Computer Society, New York. pp 498–499
11. Aumueller D, Do H-H, Massmann S, Rahm E (2005) Schema and ontology matching with COMA++. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. Association for Computing Machinery, New York. pp 906–908
12. Peukert E, Eberius J, Rahm E (2011) AMC - a framework for modelling and comparing matching systems as matching processes. In: Proceedings of the IEEE 27th International Conf on Data Engineering. IEEE Computer Society, New York. pp 1304–1307
13. Cruz IF, Antonelli FP, Stroe C (2009) Agreementmaker: efficient matching for large real-world schemas and ontologies. Proc VLDB Endowment 2(2):1586–1589
14. Duchateau F, Coletta R, Bellahsene Z, Miller RJ (2009) (Not) yet another matcher. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management. Association for Computing Machinery, New York. pp 1537–1540
15. de Carvalho MG, Laender AHF, Gonçalves MA, da Silva AS (2013) An evolutionary approach to complex schema matching. Inf Syst 38(3):302–316
16. He B, Chang KC-C (2003) Statistical schema matching across web query interfaces. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. Association for Computing Machinery, New York. pp 217–228
17. Madhavan J, Bernstein PA, Doan A, Halevy A (2005) Corpus-based schema matching. In: Proceedings of the 21st International Conference on Data Engineering. IEEE Computer Society, New York. pp 57–68
18. Su W, Wang J, Lochovsky F (2006) Holistic schema matching for web query interfaces. In: Proceedings of the 10th International Conference on Advances in Database Technology. Association for Computing Machinery, New York. pp 77–94
19. Nguyen H, Fuxman A, Paparizos S, Freire J, Agrawal R (2011) Synthesizing products for online catalogs. Proc VLDB Endowment 4(7):409–418
20. Toan NT, Phan TC, Thang DC, Hung NQV, Stantic B (2018) Bootstrapping uncertainty in schema covering. In: Proceedings of the 29th Australasian Database Conference on Databases Theory and Applications. Springer, New York. pp 336–342
21. Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. VLDB J 10(4):334–350
22. Bellahsene Z, Bonifati A, Rahm E (eds) (2011) Schema matching and mapping. Springer, New York
23. Gal A (2006) Why is schema matching tough and what can we do about it?. SIGMOD Rec 35(4):2–5
24. Wagner RA, Fischer MJ (1974) The string-to-string correction problem. J ACM 21(1):168–173
25. Miller FP, Vandome AF, McBrewster J (2009) Levenshtein distance. VDM Publishing, Saarbrucken
26. Shiang W-J, Chen H-C, Rau H (2008) An intelligent matcher for schema mapping problem. In: Proceedings of the 2008 International Conference on Machine Learning and Cybernetics. IEEE Computer Society, New York. pp 3172–3177
27. Lee Y, Sayyadian M, Doan A, Rosenthal AS (2007) etuner: tuning schema matching software using synthetic scenarios. VLDB J 16(1):97–122
28. Gal A, Sagi T (2010) Tuning the ensemble selection process of schema matchers. Inf Syst 35(8):845–859
29. Rodrigues D, da Silva AS, Rodrigues R, dos Santos E (2015) Using active learning techniques for improving database schema matching methods. In: Proceedings of the 2015 International Joint Conference on Neural Networks. IEEE Computer Society, New York. pp 1–8
30. Doan A, Domingos P, Levy A (2000) Learning source descriptions for data integration. In: Proceedings of the 3rd International Workshop on the Web and Databases. Association for Computing Machinery, New York. pp 81–86
31. Ngo D, Bellahsene Z (2012) Yam++: a multi-strategy based approach for ontology matching task. In: Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management. Association for Computing Machinery, New York. pp 421–425
32. Cudré-Mauroux P, Aberer K, Feher A (2006) Probabilistic message passing in peer data management systems. In: Proceedings of the IEEE 30th International Conference on Data Engineering. IEEE Computer Society, New York. p 41
33. Hung NQV, Tam NT, Miklós Z, Aberer K (2013) On leveraging crowdsourcing techniques for schema matching networks. In: Proceedings of the 18th International Conference Database Systems for Advanced Applications. pp 139–154
34. Aberer K, Cudre-Mauroux P, Hauswirth M (2003) Start making sense: the chatty web approach for global semantic agreements. J Web Semant 1(1):89–114

35. Alani H, Saad S (2017) Int J Adv Sci Eng Inf Technol 7(5):1790–1797
36. Duchateau F, Bellahsène Z, Hunt E (2007) Xbenchmatch: a benchmark for xml schema matching tools. In: Proceedings of the 33rd International Conference on Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco. pp 1318–1321
37. Nguyen HQV, Luong XH, Miklós Z, Quan TT, Aberer K (2013) Collaborative schema matching reconciliation. In: Proceddings of the OTM 2013 Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE. Elsevier, Amsterdam. pp 222–240
38. Duchateau F, Bellahsene Z, Coletta R (2008) A flexible approach for planning schema matching algorithms. In: Proceedings of the OTM 2008 Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE. Springer, New York. pp 249–264
39. Bishop CM (2006) Pattern recognition and machine learning (Information Science and Statistics). Springer, Secaucus
40. Rong S, Niu X, Xiang EW, Wang H, Yang Q, Yu Y (2012) A machine learning approach for instance matching based on similarity metrics. In: Proceedings of the 11th International Semantic Web Conference. Elsevier, Amsterdam. pp 460–475
41. Reis DG, Carvalho RN, Carvalho RS, Ladeira M (2017) Two-phase parallel learning to identify similar structures among relational databases. In: Proceedings of the 16th IEEE International Conference on Machine Learning and Applications. IEEE Computer Society, New York. pp 1020–1023
42. Duchateau F, Bellahsene Z (2010) Measuring the quality of an integrated schema. In: Proceedings of the 29th International Conference on Conceptual Modelling. Elsevier, Amsterdam. pp 261–273
43. Drumm C, Schmitt M, Do H-H, Rahm E (2007) Quickmig: automatic schema matching for data migration projects. In: Proceedings of the 16th ACM Conference on Conference on Information and Knowledge Management. Association for Computing Machinery, New York. pp 107–116
44. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. SIGKDD Explor Newsl 11(1):10–18

Publisher's Note