UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM

INSTITUTO DE COMPUTAÇÃO - ICOMP

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

DIEGO DE AZEVEDO RODRIGUES

A STUDY ON MACHINE LEARNING TECHNIQUES FOR THE SCHEMA MATCHING NETWORKS PROBLEM

Manaus, AM

2018

DIEGO DE AZEVEDO RODRIGUES

A STUDY ON MACHINE LEARNING TECHNIQUES FOR THE SCHEMA MATCHING NETWORKS PROBLEM

Tese apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Amazonas, como requisito parcial para a obtenção do grau de Doutor em Informática.

Orientador: Prof. D.Sc. Altigran Soares da Silva

Manaus, AM

2018

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

R696s

Rodrigues, Diego de Azevedo
A Study on Machine Learning Techniques for the Schema Matching Networks Problem / Diego de Azevedo Rodrigues. 2018
109 f.: il. color; 31 cm.

Orientador: Altigran Soares da Silva
Tese (Doutorado em Informática) - Universidade Federal do Amazonas.

1. Casamento de Esquemas em Rede. 2. Reconciliação de Esquemas em Rede. 3. Integração de Dados. 4. Aprendizagem de Máquina. 5. Banco de Dados. I. Silva, Altigran Soares da II. Universidade Federal do Amazonas III. Título

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UFAM

FOLHA DE APROVAÇÃO

"A STUDY ON MACHINE LEARNING TECHNIQUES FOR THE SCHEMA MATCHING NETWORKS PROBLEM"

DIEGO DE AZEVEDO RODRIGUES

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Altigran Soares da Silva - PRESIDENTE

Prof. João Marcos Bastos Cavalcanti - MEMBRO EXTERNO

Profa. Ana Carolina Brandão Salgado - MEMBRO EXTERNO

Profa. Carmem Satie Hara - MEMBRO EXTERNO

Prof. José Antônio Fernandes de Macêdo - MEMBRO EXTERNO

Manaus, 22 de Outubro de 2018

Resumo

Casamento de Esquemas é a tarefa de encontrar correpondências entre elementos de diferentes esquemas de bancos de dados. É um problema desafiador, uma vez que o mesmo conceito geralmente é representado de maneiras distintas nos esquemas.Tradicionalmente, a tarefa envolve um par de esquemas a serem mapeados. Entretanto, houve um crescimento na necessidade de mapear vários esquemas ao mesmo tempo, tarefa conhecida como *Casamento de Esquemas em Rede*, onde o objetivo é identificar elementos de vários esquemas que correspondem ao mesmo conceito. Este trabalho propõe uma família de métodos para o problema do casamento de esquemas em rede baseados em aprendizagem de máquina, que provou ser uma alternativa viável para o problema do casamento tradicional em diversos domínios. Para superar o obstáculo de obter bastantes instâncias de treino, também é proposta uma técnica de *bootstrapping* para gerar treino automático. Além disso, o trabalho considera restrições de integridade que ajudam a nortear o processo de casamento em rede. Este trabalho também propõe uma estratégia para receber avaliações do usuário, com o propósito de melhorar o resultado final. Experimentos mostram que o método proposto supera outros métodos comparados alcançando valor *F1* até 0.83 e sem utilizar muitas avaliações do usuário.

Palavras-chave: Casamento de Esquemas em Rede, Reconciliação de Esquemas em Rede, Integração de Dados, Aprendizagem de Máquina, Banco de Dados.

Abstract

Schema Matching is the problem of finding semantic correspondences between elements from different schemas. This is a challenging problem, since the same concept is often represented by disparate elements in the schemas. The traditional instances of this problem involved a pair of schemas to be matched. However, recently there has been a increasing interest in matching several related schemas at once, a problem known as *Schema Matching Networks*, where 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 automatically generate training data. In addition, we leverage constraints that arise in network scenarios to improve the quality of this data. We also propose a strategy for receiving user feedback to assert some of the matchings generated, and, relying on this feedback, improving the quality of the final result. Our experiments show that our methods can outperform baselines reaching F1-score up to 0.83.

Keywords: Schema Matching Networks, Schema Reconciliation Networks, Data Integration, Machine Learning, Databases.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last decade, more and more applications and systems have been built to produce information, which are usually stored in databases. Hence, there has been an increasing demand to enable different databases, e.g., maintained by different corporations/institutions, on a given application domain to be integrated or at least interoperate. As these databases are designed by different people, they often follow distinct logical designs, i.e., the same information is represented in different ways in each one of them.

For instance, different universities maintain their own databases of academic information, each one with a specific logical design. To get global statistics of these universities, one needs to integrate data from these databases, which store data in the same domain, but have different schemas. The integration process requires that one should know which elements in each schema represent the same concept in the real world. This is called the *schema matching* task [Bonifati and Velegrakis, 2011]. Once the matching is complete, further operations can be performed such as comparisons, union, concatenation, etc.

This *matching* task is the first challenge in every integration process and it is crucial that it is correctly performed in order to allow the interoperability between systems. *Schema matching* is the task of finding semantic correspondences between elements (or attributes) of two given database schemas [Do and Rahm, 2002, Madhavan et al., 2001, Doan et al., 2001, Bernstein et al., 2011, Bonifati and Velegrakis, 2011]. Such task is very important for enabling data integration and interoperability in domains such as e-commerce, geospace, biology, health, etc.

In this Chapter, we present the schema matching problem through a motivating example and we present a glimpse of work done addressing this task. We also address how approaches started considering matching a network of many schemas and the new challenges the task presented. Next, we introduce this thesis goals and research questions related. Finally, we summarize our contributions and present the thesis organization.

1.1 Motivation

Consider the (simplified) schemas of databases on academic information from univerisities illustrated in Figure 1.1. The schema matching task is to identify the matchings (depicted as dotted lines) between elements from those schemas. In this example, the schemas model data on students in different ways, even using particular representations of hierarchy and attribute names. Schema attributes may represent the same concept in the real world. However they are often modeled using different attribute names. For instance, `BROWN.PERSONALDATA.NAME.Firstname` and `CMU.biography.fname` represent the

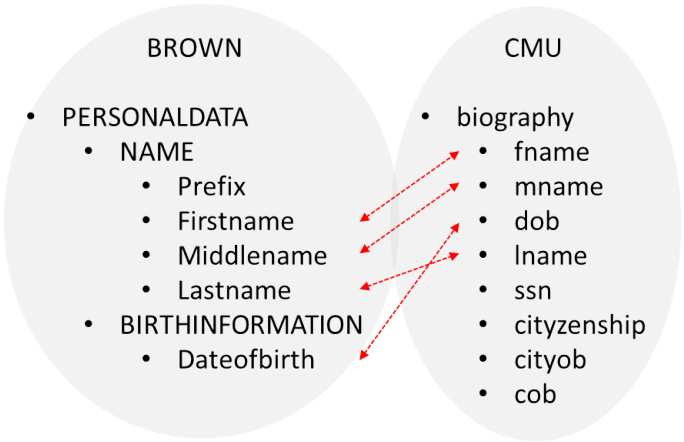

Figure 1.1: Parts of the schemas from university application forms (Brown and CMU) and matchings between elements (dotted lines).

student's first name. Also, there might be more challenging matchings to be discovered, e.g. `BROWN.BIRTHINFORMATION.Dateofbirth` and `CMU.dob`, both representing the students date of birth, which may be validated only by a domain specialist or the schema owners.

Traditionally, the schema matching task is performed manually by specialists who have extensive knowledge about the domain and the schemas involved. However, even for a specialist, this task may be time-consuming, costly and error-prone.

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. In addition, 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 that is represented by a single element in the other schema.

Over the years, several research initiatives have been carried out

to deal with schema matching, resulting in a number of papers published [Madhavan et al., 2001, Doan et al., 2012, Do and Rahm, 2002, Melnik et al., 2002, Li et al., 2005, Hung et al., 2014] and several prototypes and commercial systems were made available [Popa et al., 2002, Aumueller et al., 2005, Peukert et al., 2011].

Many of these methods rely on a set of predefined steps and parameters to perform the matching between schemas (e.g. [Do and Rahm, 2002, Aumueller et al., 2005, Cruz et al., 2009]). While these methods perform well in certain domains, their performance decreases when applied in adverse scenarios, as they need tuning on strategies and parameters.

As an alternative to fixed heuristics methods, machine learning methods arose with the advantage of creating a specific model for each matching scenario (e.g. [Doan et al., 2001, Duchateau et al., 2009, de Carvalho et al., 2013]). These methods are easy to adapt to different matching tasks, but they require the user to label a large amount of examples to train its models which is a disadvantage compared to heuristic methods in terms of user-effort.

As the problem evolved, schema matching tasks have appeared in a networked scenario, where more than two data sources (schemas, query forms, databases) need to be matched [He and Chang, 2003, Madhavan et al., 2005, Su et al., 2006, Nguyen et al., 2011, Hung et al., 2014, Toan et al., 2018]. In this case, the matching task is performed with all the schemas together, instead of between pairs of schemas as depicted in Figure 1.2. From this point forward, the matching between only two schemas will be referred as the *classic schema matching* and, when the task involves more than two schemas, it will be referenced as the *schema matching networks* task.

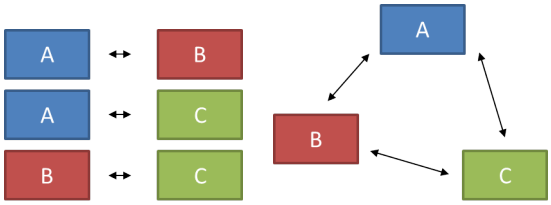

Figure 1.2: Classic Schema Matching (1.2(a)) only takes two input schemas while in the Schema Matching Network (1.2(b)) setting all the schemas are matched together.

In the classic schema matching, the methods are usually executed by submitting all the combinations of schemas involved. For instance, schema `A` with schema `B`, schema `A` with schema `C` and schema `B` with schema `C`. In the schema matching networks task, the method receives all the schemas together and performs multiple matchings at once. This setting yields the use of extra information based on integrity constraints between schema elements, as it is done in methods proposed for similar problems [Aberer et al., 2003, Cudré-Mauroux et al., 2006], and takes advantage of similar concepts that are present in a network [Su et al., 2006, Madhavan et al., 2005, Nguyen et al., 2011].

In Figure 1.3 we illustrate the result of a schema maching networks task. Each point represents an schema element, points closer to one another mean they come from the same schema. A line between two points means they were matched, therefore, they have the same meaning in the real world. Points without lines could not be matched to elements in other schemas, whether the method used could not find any correspondences or they actually do not have any correspondences in other schemas.

The schema matching networks task shares some of the same challenges

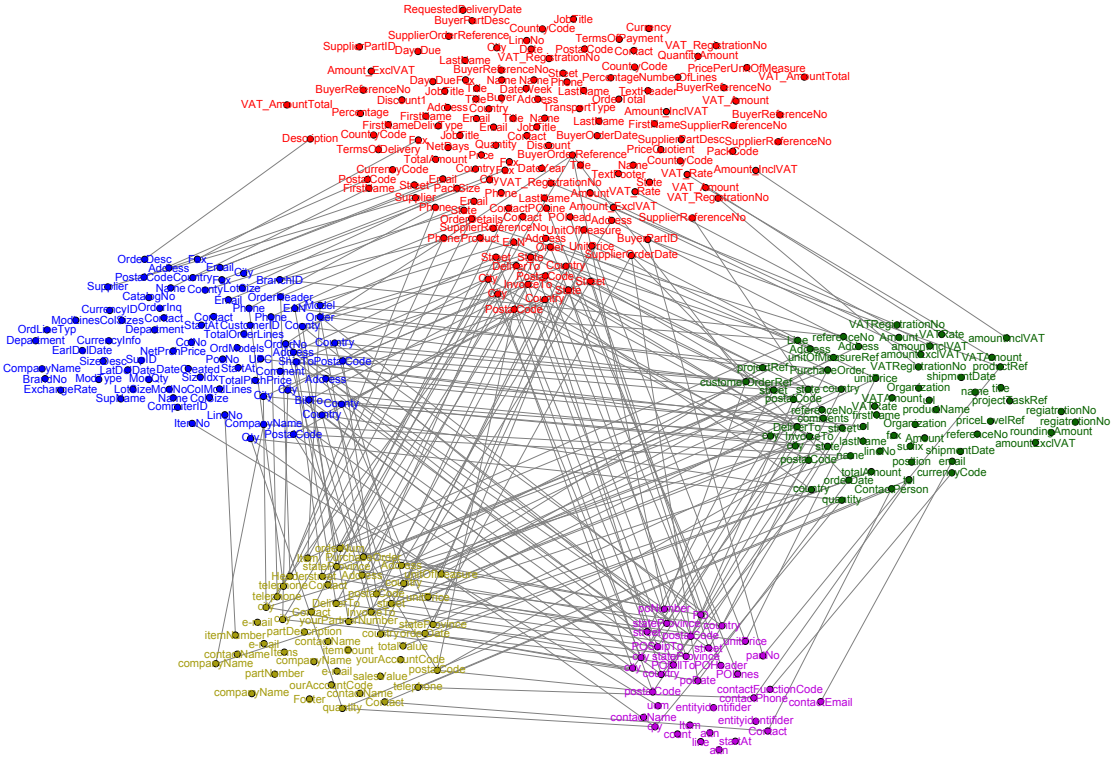

Figure 1.3: Resulting matching network of the Purchase Order dataset.

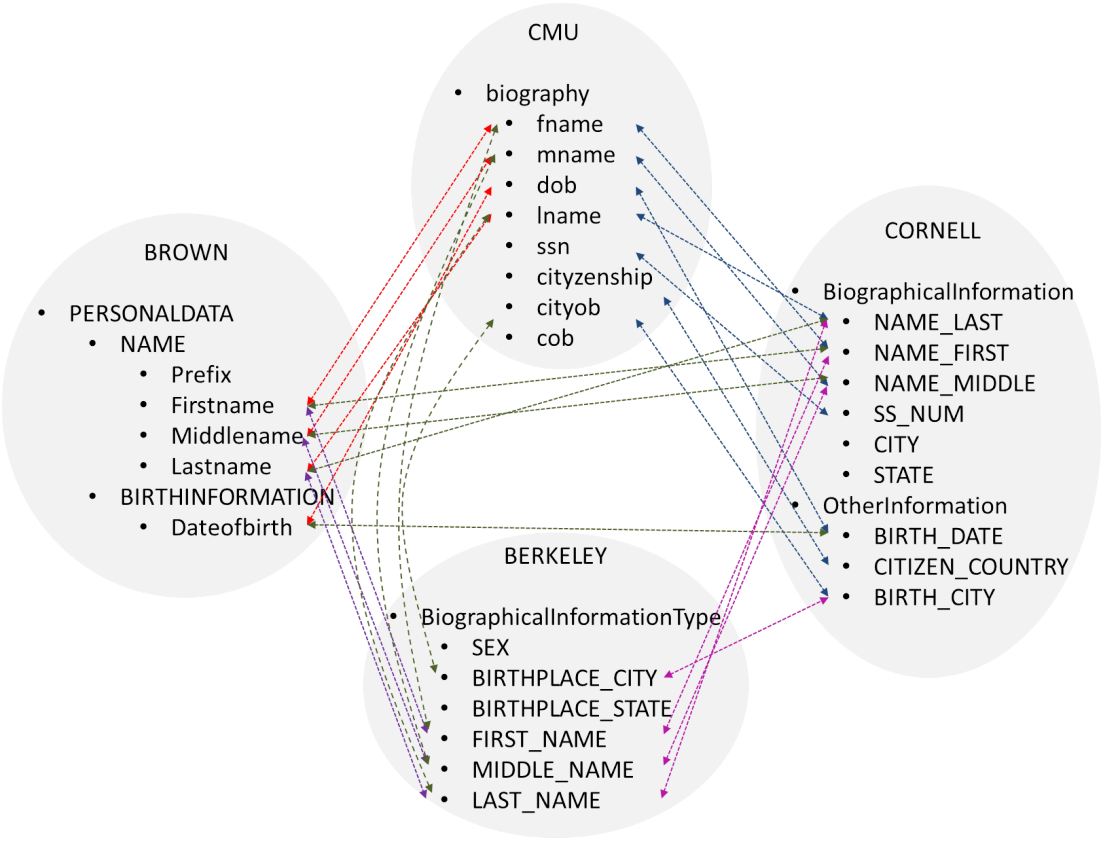

Figure 1.4: Schema matching networks task: the schemas from university application forms (CMU, Brown, Cornell, and Berkeley) and matchings between elements (dotted lines).

as the classic setting. Consider as an example the matching network in Figure 1.4. Schema elements might have slight different names besides model the same concept, such as `BROWN.PERSONALDATA.NAME.Lastname` and `CORNELL.BibliographicInformation.NAME_LAST` meaning the last name of a student, elements with similar names refer to distinct concepts, e.g. `CMU.cityob` referring to *city of birth* and `CORNELL.CITY` referring to a student's *current city*, and attributes with different structures representing the same concept (`BROWN.PERSONALDATA.NAME.Firstname` has path with depth = 4 and `BERKELEY.BiographicalInformationType.FIRST_NAME` has path with depth = 3, the inner elements do not share similar words).

The schema matching networks task also introduces a new aspect to consider: the integrity of the matching network. To guarantee that matchings across the network remain consistent, several *network constraints* have to be taken into consideration when making matches between elements. One of the constraints to be considered is the *cycle constraint*: matched elements from different schemas should form a cycle in the graph formed by all the schema elements. Consider the network in Figure 1.5 and that the element `BROWN.PERSONALDATA.NAME.Firstname` is matched to both `CMU.biography.fname` and `CORNELL.BiographicalInformation.NAME_FIRST`. For the cycle constraint to be obeyed, we should add the matching between `CMU.biography.fname` and `CORNELL.BiographicalInformation.NAME_FIRST`.

Besides that, schema matching networks methods still have to cope with the number of schemas presented. The more schemas involved, more schema elements to be matched and the number of possible matching combinations grows exponentially, which can be overwhelming.

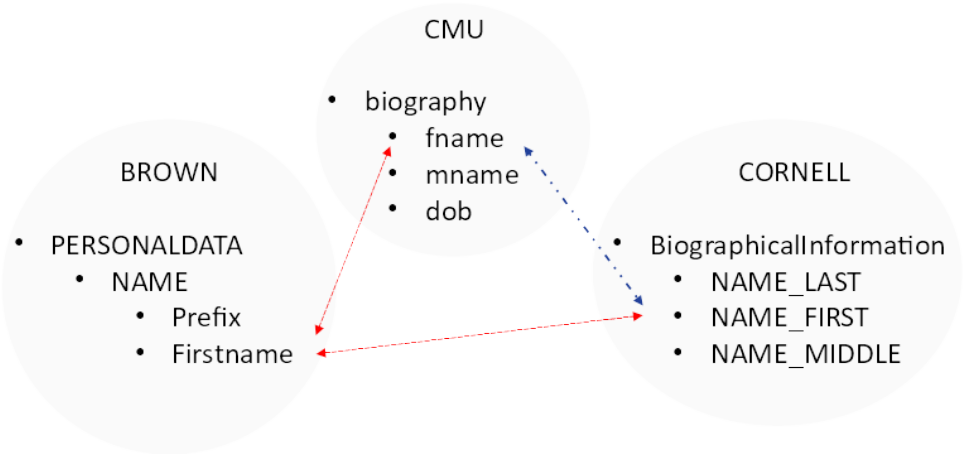

Figure 1.5: The cycle constraint requires that matched elements from different schemas should form a cycle in the graph.

We recognize that matching tasks can involve *complex matches*. They occur when *one* element from a schema can be matched to *many* elements in other schema, such as `Fullname` could be matched to `FirstName`, `MiddleName` and `LastName`. Complex matches can make the problem even more challenging and there are studies devoted only to this kind of matching [de Carvalho et al., 2013, He et al., 2004]. Due to its incompatibility with the network constraints (presented later in Section 2.2.2), we consider these kind of matchings out-of-scope and do not address them in our work.

In spite of being a successful approach to the classic matching pairs of schemas, only recently machine learning techniques have been considered for the network scenario. For instance, the work presented in [Hung et al., 2014] applies a boosting technique, by querying the user for accepting or rejecting correspondences given by a matching method while maintaining a network graph with probabilities of the matching correctness.

1.2 Goals and Research Questions

Our work focus on studying new methods for the schema matching networks task through machine learning. Our main goal is to use machine learning techniques to address schema matching network tasks and produce good quality matching results. We also consider the user participation in the process and aim to easier its effort during the task execution. Hence, we define research questions that drive this work:

- *Can machine learning methods be suitable when considering the schema matching networks scenario?* Machine learning methods can achieve good results in the classic schema matching problem, if the training is done correctly, this approach can create different models for each matching tasks and be independent of expert tuning.

- *How to acquire a large number of training examples without asking the user for labels? Can we generate good models with this training?* Machine learning based methods depend directly of the training examples provided, regarding both quantity and quality. We study how these two factors affect the quality of models learned and matchings generated.

- *Can we increase matching quality by asking the user for approval of correspondences? Can we promote this task without causing the exhaustion of the user by asking a large number of assertions?* Usually the user participates in the matching process by reviewing the correspondences generated by a method. We study how much the matching quality is increased when the user reviews answers. Also, we study if techniques can be applied to reduce

the user effort in the reconciliation task while maintaining the quality of the matchings.

1.3 Contributions

We make our contibutions through three methods in Chapter 3. We summarize them in the following list:

- First, we investigate an approach in which classifiers are trained with existing matchings given by a specialist. In the experiments, we show that our method reaches high values of precision, for instance 0.89 in one of the matching tasks, topping the baselines. Also, to the best of our knowledge, this is the first supervised learning method that uses classifiers to address the schema matching networks problem.

- Next, we investigate if machine learning methods can cope with several different domains in opposition to heuristic methods. Heuristic methods need a different set of parameters to achieve the best results in every different matching task. As the schema matching networks task can be viewed as an aggregation of various classic matching tasks, to find a common good set of parameters might require a lot of knowledge of the method and the schemas domain. This challenge may be better addressed by using a learning strategy, as it can create models independent of domain as long as it has matching examples. In our experiments, we show that while the heuristic methods, depending on the domain, perform better in favor of either *precision* or *recall*, the more stable balancing of the learning strategy leads to better results in

the average scenario.

- Then, as labeled examples may be hard to obtain, we also present a follow-up method that removes the requirement of having previously labeled examples. We rely on examples from previous matchings obtained by heuristic unsupervised methods and use network constraints to prune mislabels. Hence, we guarantee a large training set of examples with no cost to the user, saving them for the last stage of the integration task, where she can correct mislabels. In experiments, we show that `RF4SM-B` (*Random Forest for Schema Matching - Boosting*) at least matches the F1-score of `RF4SM` besides using no previously labeled examples. We also show that, on average, it outperforms the baseline methods by 0.4 in F1-score.

- Finally, the matching process ends with a *reconciliation* task, in which a specialist reviews matches generated by the matching method. Hence, we also address this last step in the matching process by taking both user-input and the network constraints to enhance matching quality. By doing that, we show the virtually maximum level of F1-score the methods can reach. In our experiments, we show that our method allows the specialist to participate in the process by reviewing at least 4 times less instances than the baselines and helping the method to achieve even better matching quality.

This thesis proposal is organized as follows:

- Chapter 2 presents the necessary background on the schema matching problem and reviews some of the related work on schema matching, including heuristic methods, machine learning methods and works that consider the schema matching networks scenario.

- Chapter 3 presents, first, RF4SM (*Random Forest for Schema Matching*) – the supervised method based on classifiers that addresses the schema matching networks problem. Then, we present RF4SM-B (*Random Forest for Schema Matching - Boosting*) – the boosting strategy that removes the predefined training examples and gathers automatic matchings from heuristic methods and uses them as training, and finally, we show RF4SM-B-Rec (*Random Forest for Schema Matching - Reconciliation*) – the strategy which brings the user back in the process making them review matchings.

- Chapter 4 presents the experimental evaluation performed and discussions of results observed.

- Chapter 5 offers conclusions and shows future directions of our work.

Chapter 2

Background and Related Work

In this chapter we review basic concepts of the schema matching problem by showing examples and the literature that we consider as more related to our work. We review the work addressing the classic schema matching as well as the schema matching networks problem. We also address the work that uses network constraints in similar problems and how the constraints were used. Finally, we review some work adressing the reconciliation of matching networks. A more comprehensive coverage on the general schema matching problem, which has been explored for a longtime in the literature, can be found in several surveys and text books [Rahm and Bernstein, 2001, Doan et al., 2012, Bellahsene et al., 2011, Bernstein et al., 2011]. As for the other problems addressed, we highlight some of the projects that stood out in the literature over the years.

2.1 Classic Schema Matching Problem

Schema matching is the task of finding semantic correspondences between elements of two schemas [Do and Rahm, 2002]. The schemas can be any distinct heterogeneous data sources (e.g. database schemata, XML DTDs, HTML form tags, etc.) in the same domain [Gal, 2006]. As it results in connecting two different sources of information, the task is considered to be required in any data integration task.

Despite several prototypes and methods being presented over the years addressing this task, there is no method considered to completely solve the problem. In addition, often a specialist user is required to review answers after a method is executed, to guarantee the quality of matching results.

Usually, schema matching methods apply one or more functions to establish a similarity value between pairs of elements from schemas. Each pair of elements is called a *matching candidate*. These functions, called *matchers*, receive two elements as input and estimate a similarity value between 0 and 1, the highest the value more similar the elements are. Figure 2.1 shows an example of similarity values generated by the well-known Levenshtein distance function [Miller et al., 2009] calculated for some elements from two datasets in the Books domain.

Matchers can apply several techniques to measure similarity between elements. These techniques can be based on the characteristics of schema elements, structural information, constraints and rules designed by specialists [Nguyen, 2014]. They apply functions on characteristics such as searching for commom prefix/suffix in names, compare n-grams, use external information such as dictionaries, compare the structure of the schemas (tree heights, leaves and children, etc.) and get hints provided by users.

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

Figure 2.1: Similarity values based on the Levenshtein distance.

Many methods usually have a library of matchers available. After calculating similarities of matching candidates, these methods might aggregate all similarities in one matrix (i.e. by calculating the average of all similarities) and use it to select the matches. The selection of matches can be carried out with simple strategies such as applying a threshold, or more sophisticated techniques such as machine learning.

In the example of Figure 2.1, 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`⟩.

In the following sections, we highlight some works addressing the classic schema matching problem. They may rely on heuristics or machine learning techniques. While heuristics are faster and usually produce a large number of matching, machine learning methods can produce dedicated models for each different task and generate better matching results.

2.1.1 Heuristic Methods

The heuristic methods are largely used as they are simple to implement/ execute, often run fast and generate a large number of matches. However, these methods are not always consistent when running through different datasets. As observed by previous studies [Rahm and Bernstein, 2001, Bernstein et al., 2011], these methods can perform better depending on the dataset and the parameteres chosen. Some systems such as eTuner [Lee et al., 2007] and SMB [Gal and Sagi, 2010] were devoted to determine how the matching can be improved by tuning parameters. Systems such as Similarity Flooding [Melnik et al., 2002],

CUPID [Madhavan et al., 2001], COMA [Do and Rahm, 2002] and Agreement Maker [Cruz et al., 2009] are representative systems that use fixed heuristics for combining matchers.

`COMA/COMA++` [Do and Rahm, 2002, Aumueller et al., 2005] is a method for *co*mbining *m*atching *a*lgorithms. Its execution is a good illustrator of how heuristic methods perfom. Its flow of execution is depicted in Figure 2.2: (a) it starts by receiving two input schemas from the same domain; (b) all combinations of elements from the schemas are submitted to pairwise functions called *matchers* (such as the *Levenshtein distance* or evaluating the compatibility of elements *data types*). The matchers assign a similarity value to each pair of elements, one of each schema, according to a predefined function that ranges between [0,1], being 1 the similarity score that means a high similarity. After calculating the similarities, one aggregator function (such as the *Average*) is used to summarize all the similarities calculated by matchers into one matrix; (c) finally, it applies a selection method (such as using *thresholds*) to generate the method's matching answers and (d) present them to user.

The authors report, empirically, the best combinations of strategies which are in its majority, lexical functions. In fact, it is worth noting that a subset of the matchers proposed is reported as leading 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 results for different configurations of `COMA` and argue that each different configuration can be used in different scenarios. When its best configuration (also referred as the default configuration) is used, `COMA`

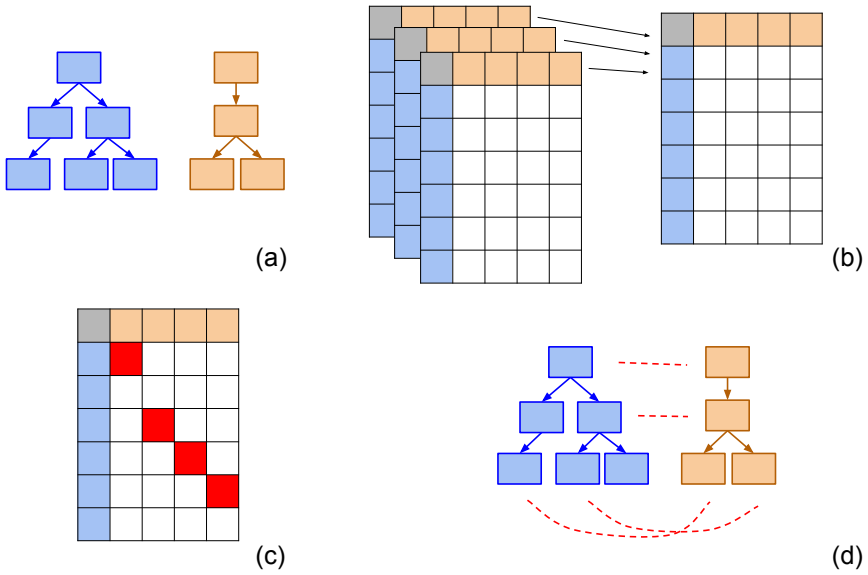

Figure 2.2: General process of COMA/COMA++ execution.

achieved Overall[1] (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` [Melnik et al., 2002] appears as an alternative that strongly considers the structural aspect of the schemas as its algorithm is based in graph analisys. Its execution is depicted in Figure 2.3.

Similarity Flooding algorithm starts by (a) transformating the two input schemas in graphs. Then it applies (b) a string matcher to estimate initial similarities between the pairs of elements from the schemas. Next, (c) the flooding algorithm propagates similarities through graph nodes. Similarities between elements in the same branches tend to receive partial scores from ancestor elements;

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

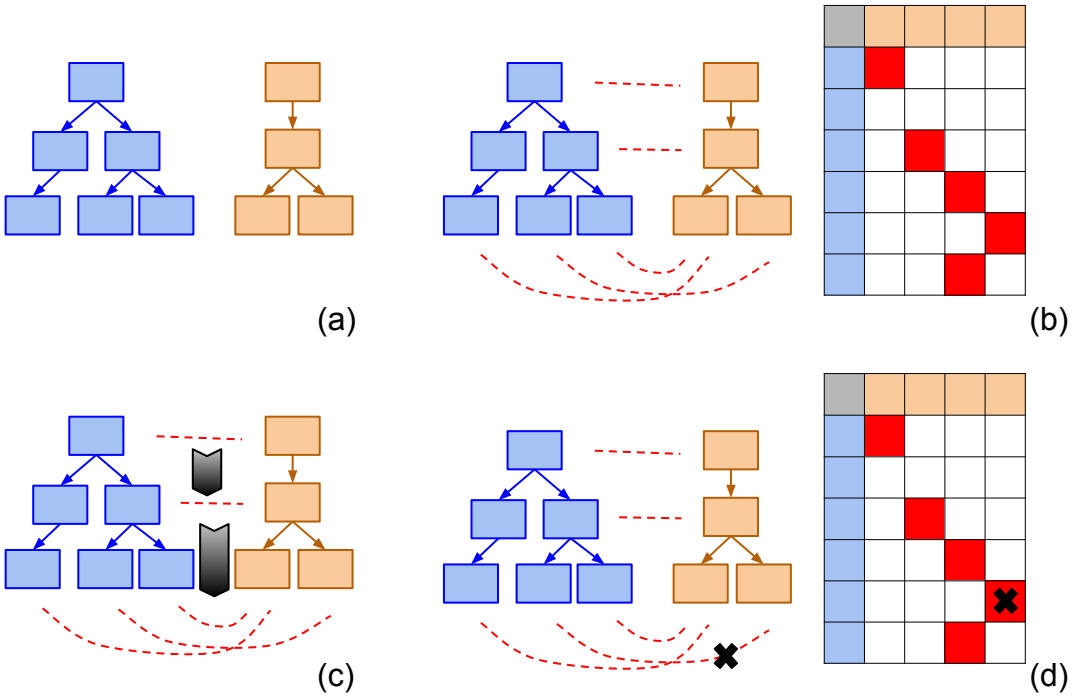

Figure 2.3: General process of Similarity Flooding execution.

Finally, (d) a threshold is applied to prune the most 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 were provided by volunteers. On average, the Similarity Flooding method reached an accuracy value around 0.55. The authors note that in some of the tasks, the algorithm could perform better as the schemas had more structural information and, when there were zero or minimal structural information, the method would struggle to find correspondences. The authors also conclude that for many matching tasks, at least half of the manual work in matching two schemas can be saved by using their algorithm. The work by [Shiang et al., 2008] also uses graph similarity, but they use a string matcher to filter matching candidates and to reduce computation. The method was evaluated with nine matching

problems and the authors report on average accuracy around 0.59.

As implementations of the methods can be found publicly, COMA and Similarity Flooding were used as representatives of heuristic methods in our experiments. We give a brief description of two more representative systems that use this kind of strategy: CUPID and Agreement Maker.

In CUPID [Madhavan et al., 2001], the authors propose a combination of two types of matchers: linguistic matchers and structural matchers. Linguistic matchers use element names, data types and domains. A thesaurus is adopted to address short forms, acronyms, and synonyms. Structural matchers are based on semantic relationships of schema elements, e.g., elements with matched sub-elements in XML schemas, are matched. To determine mappings, the similarity evaluations provided by the matchers are combined using a weighted mean, with weights ascertained manually. The authors compare CUPID with other matching systems according to different aspects such as the use of schema structure, thesaurus and user interaction, though they do not report results on the quality of the results generated (e.g, precision and recall).

Agreement Maker [Cruz et al., 2009] uses a library of matchers in layers. The result of the first layer can be used as the input of the following layers. The matching process is divided into two steps: similarity computation and mapping selection. In the first step, each pair of elements from both schemas is compared in order to generate similarities matrices. In the second step, the similarity matrices are scanned to select the best matchings, taking into account thresholds and the cardinality of correspondences. Authors presented a demo [Cruz et al., 2009] but did not report experimental results.

2.1.2 Machine Learning Methods

Distinct methods can apply a number of different techniques to modify and select similarity values from the matrices. Some techniques can be heuristics (as highlighted in the previous section) or machine learning driven approaches. The latter is one of the most popular approaches, transforms the similarity matrices in a list of matching candidates with a set of characteristics (i.e. their similarity values as depicted in Figure 2.4) and addresses the problem as a machine learning classification problem, in which instances must be classified as `TRUE` if they are a true matching pair or `FALSE` otherwise.

In the example of Figure 2.4, each line of the table (right side) is a matching candidate, containing two schema elements and their similarity values. Each matching candidate must receive a label `TRUE` or `FALSE` depending or whether or not they represent the same concept.

Once the matching task is modeled as a classification problem, any machine learning classification algorithm can be, in principle, used to obtain matches. Strategies vary from learning weights to similarity matrices, to complex models such as neural networks, and to more intuitive and readable strategies such as decision trees. As any learning-based strategy, learning-based schema matching methods also require labeled training examples so a model can be generated. Once a model is generated, it is used to evaluate unlabeled instances and classify them as `TRUE`, if they represent a matching, or `FALSE`, otherwise.

In Figure 2.5, a model based on decision tree is depicted. Decision Tree [Quinlan, 1993] is a simple learning strategy that takes decisions in its inner nodes based on the evaluated instance's features. When the decisions reach a leaf node,

Similarity Matrices

Levenshtein Matcher		Schema B			
		searchForm .search	searchForm. searchType	searchForm. keyword	...
Schema A	search	1.0000	0.6000	0.1429	...
	search.field-title	0.0909	0.1818	0.0909	...
	search.field-subject	0.1538	0.1538	0.0769	...
	...	...	...	...	...

Children Matcher		Schema B			
		searchForm .search	searchForm. searchType	searchForm. keyword	...
Schema A	search	0.0385	0.0421	0.0466	...
	search.field-title	0.1850	0.2511	0.1850	...
	search.field-subject	0.2750	0.2333	0.3292	...
	...	...	...	...	...

Machine Learning Table

Schema A	Schema B	Levenshtein	Children	...
search	searchForm.search	1.0000	0.0385	...
search	searchForm.searchType	0.6000	0.0421	...
search	searchForm.keyword	0.1429	0.0466	...
search.field-title	searchForm.search	0.0909	0.1850	...
search.field-title	searchForm.searchType	0.1818	0.2511	...
search.field-title	searchForm.keyword	0.0909	0.1850	...
search.field-subject	searchForm.search	0.1538	0.2750	...
search.field-subject	searchForm.searchType	0.1538	0.2333	...
search.field-subject	searchForm.keyword	0.0769	0.3292	...
...	...	....	...	...

Figure 2.4: Changing from similarity matrices to machine learning instances.

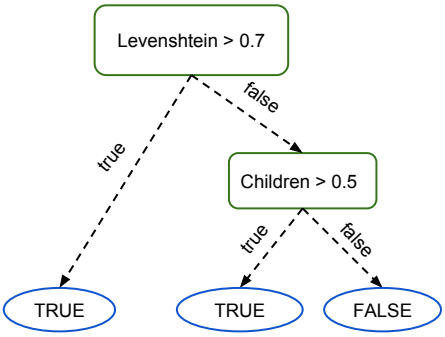

Figure 2.5: A single decision tree can classify pairs based on their similarity values.

the decision tree gives its classification for that instance.

Taking as example the matching candidate ⟨`search`,`searchForm.search`⟩ described as the first row in the table in Figure 2.4, the tree would first evaluate the decision in the root node: as the matching candidate Levenshtein value is higher than 0.7, the tree would follow its left path leading to the leaf node `TRUE`, which is the label given to that candidate by that decision tree.

The next matching candidate ⟨`search`,`searchForm.searchType`⟩ when submitted to the same decision tree to evaluation would take the right path after the first decision (as the matching candidate value of Levenshtein is lower than 0.7), the second decision also leads to the right path (as the value of Children is lower than 0.5), and finally reaches a leaf node making the label definition as a `FALSE` matching.

As using one single decision tree is a very brittle approach, proposed methods usually combine many learning methods to obtain better matching quality [Ngo and Bellahsene, 2012]. State-of-the-art methods use several strategies, such as combining several decision trees to build a

meta-model [Duchateau et al., 2009, Rodrigues et al., 2015], combining different learning algorithms to create more complex models [Doan et al., 2000, Gal and Sagi, 2010], and using specialists inputs to help learning models to better weigh strategies and tune learning parameters [Li et al., 2005].

Once machine learning classifiers require a training set, the user must classify a substantial amount of instances in the same domain, which may be hard to acquire. This is a disadvantage if we compare them to the previously described heuristic methods, which are unsupervised. On the other hand, machine learning-based methods are able to generate different models for different matching tasks. A few methods in the literature have applied machine learning techniques to the problem of combining matchers. Amongst the most representative methods in this category we cite LSD [Doan et al., 2000], SMDD [Li et al., 2005], YAM [Duchateau et al., 2009] and SMB [Gal and Sagi, 2010].

The LSD system [Doan et al., 2000] was one of the first using a machine learning method in schema matching problems. According to this approach, first, the element-level matchers, called base learners, are evaluated using training data. Then, a component called meta-learner selects and weighs the base learners according to their performance on training data. The model resulting from this process is used for matching tasks. A number of matchers based on schema information, instance information and schema constraints are used. LSD is extensible to receive new matchers. Reported results show that LSD system achieved a predictive accuracy of from 0.62 to 0.75. All the experiments were conducted on a single dataset in the Real State domain[2].

SMDD - Schema Matching based on Data Distribution [Li et al., 2005] is a

[2]At the time of writing, we could not find a public source link containing this dataset

method based on neural networks. It analyses data contents to perform schema matching. In each matching task, the method receives two schemas S and T. The method extracts data instances from the schema elements (from schema S) and, by analyzing its distribution and characteristics, generates distribution vectors as output. Then, these vectors are clustered into categories (the number of categories is defined by the user). This result is used to train a neural network in a way that each node in the output layer represents a cluster. Learning rates can be set by the user. The distribution vectors from schema T are used as input in the network and the output is the similarity between input vectors and each category of S. Finally, candidates with high similarity are selected as answers. The experiments were conducted using relational schemas from Educational domain. Reported results of the best cases show a F-Measure value over 0.65.

YAM/YAM++ (Yet Another Matcher) [Duchateau et al., 2009, Ngo and Bellahsene, 2012] is based on the idea of generating a dedicated schema matcher for every task, i.e. it generates a different machine learning model for every new matching task. We depict its execution in Figure 2.6. It has two phases: a learning phase and a matching phase. In the learning phase, the method uses a database of matching examples to train its library of matchine learning classifiers [3]. The matching examples are acquired by saving previous matching tasks or they are provided by a user. Then, the best classifier in the learning phase is elected as the *dedicated matcher*. In the matching phase, the dedicated matcher is used to generate the correspondences of the current matching task. Optionally, the user can input correspondences as training and state a preference for precision or recall. Experiments reported show that YAM

[3]Classifiers from the Weka package available at http://www.cs.waikato.ac.nz/ ml/weka

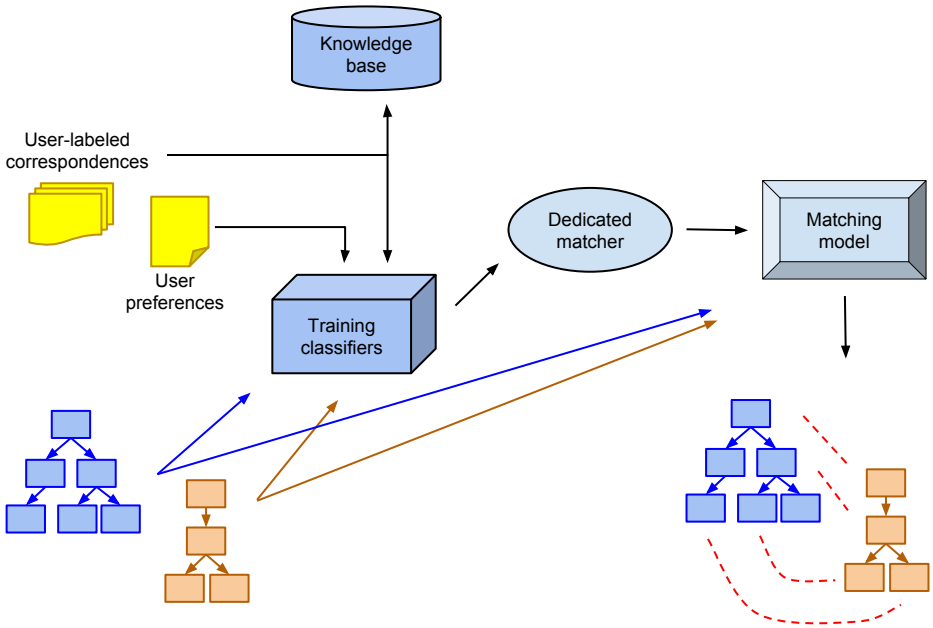

Figure 2.6: YAM first uses its knowledge base to create a dedicated matcher, then it creates correspondences between schemas.

outperformed COMA++ and Similarity Flooding. It reached an F-Measure of 0.8 in 4 out of 10 matching tasks. The experiments were conducted on several datasets including travel, currency and webform domains.

As being one of the state-of-the-art machine learning methods for the schema matching problem, YAM was chosen to be the machine learning baseline method in our experiments. However, as no public implementation of YAM is available, we use an emulation of the method in our experiments.

We highlight other methods that use machine learning to address the classic schema matching problem: SMB [Gal and Sagi, 2010] which tries to use boosting to tune matchers and the method by [de Carvalho et al., 2013] which proposes a genetic programming method to find complex matches.

In [Gal and Sagi, 2010], the authors propose a supervised method called Schema Matcher Boosting (SMB), which uses the AdaBoost algorithm to combine schema matchers. In this method, First-line matchers (which receive two attributes as input) are applied to generate similarity matrices. Second-line matchers (which receive a similarity matrix as input) are applied to transform one (or more) similarity matrices into another similarity matrix. These matchers are modeled as weak classifiers and AdaBoost is used to weigh these classifiers taking advantage of the *Boosting* technique. The authors reported results showing that SMB can improve weak classifiers F-Measure up to 34% achieving an F-Measure around 0.75.

The work by [de Carvalho et al., 2013] proposes a genetic programming method to find complex matchings between two databases. The method analyses data from two databases and apply strategies previoulsy used in record deduplication and information retrieval fields to find complex matches. The method is based on genetic programming. It considers matching functions (and its combinations) as individuals of the evolutionary population. Following the idea of genetic programming approaches, it performs genetic operations such as *crossover* and *mutation* to evolve individuals for a predefined number of rounds. At the end of the process, the population of individuals define which functions will be used to match the databases. The experiments were conducted with three synthetic datasets and one real dataset from Google Fusion Tables [4]. The reported results were similar for both real and synthetic datasets. It achieved accuracy value of 1 when identifying matchings in Real State and Restaurant datasets. The accuracy for Car Dealers dataset was 0.8. We note that, in our work, we do not consider the identification of complex matches as they do not comply with the network constraints presented

[4]https://support.google.com/fusiontables

later in Section 2.2.

Besides taking only two schemas as input, some methods in the literature often had considered using information about other schemas in the same domain, in order to achieve higher accuracy in creating matchings. Some methods would save previous matches and similarity values to build a knowledge base to train machine learning methods [Madhavan et al., 2005, Nguyen et al., 2011]. As methods evolved considering more information and using more complex approaches, the problem also evolved by the need of finding matches involving data in more than two schemas. Hence, methods started to consider the schema matching networks problem.

2.2 Schema Matching Networks Problem

With the number of different sources getting higher and the necessity of interoperability between systems being more commom, there has been a necessity of creating *global schemas* to represent many different sources in a integration process. However, when adding a new source to the process, the global schema created often needed to be rebuilt making this an impractical solution [Cudré-Mauroux et al., 2006]. 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 [Hung et al., 2013].

Classic schema matching methods can be used to generate attribute correspondences between all pairs of schemas. By combining these matches, we have a

notion of a *schema matching network*: a network of connected schemas in which two schemas to be matched do not exist in isolation but participate in a larger interaction and connect to several other schemas at the same time [Nguyen, 2014].

Consider the schema matching network presented in Figure 1.4. A classic matching method could execute runs by taking two schemas at a time (⟨Brown,CMU⟩, ⟨Brown,Cornell⟩, ⟨Brown,Berkeley⟩, ...), and after that, we could combine all matchings discovered and create a network of matches. However, due to limitations of such methods, they often generate mismatches and create incoherent matchings when evaluating the network as a whole.

To cope with such a coherence issue, some proposals in the literature considered using network integrity constraints to help maintain the consistency of a matching network [Hung et al., 2014, Hung et al., 2015].

2.2.1 Classic Schema Matching versus Schema Matching Networks

Let schemas $S = \{s_1, s_2, ..., s_n\}$ and $T = \{T_1, T_2, ..., T_m\}$ be schemas with their respective set of elements. A classic schema matching task M is a tuple $M = \langle S, T, C\rangle$ containing the schemas S and T, and a set of matching candidates $C = \{c_1, c_2, ..., c_j\}$, where each matching candidate is a tuple $c = \langle s, t, l, v\rangle, \forall s \in S, \forall t \in T$. Each matching candidate c has associated a label $l \in \{\texttt{TRUE},\texttt{FALSE}\}$ which value is `TRUE` if the elements represent a real matching or `FALSE` otherwise and a vector of similarities v.

Each matching candidate c has also associated a vector $v = \langle v_1, v_2, ...v_m\rangle (\, m \geq 1, v_m \in [\,0, 1]\,)$. Each value in v is a similarity score given by one of the m

matchers available reflecting how similar the elements from c are. Consider the values given by the matchers *Levenshtein* and *Children* in Figure 2.4, the candidate $\langle$`search.field-subject`,`searchForm.keyword`$\rangle$ has the vector $v = \{0.0769, 0.3293\}$ as its similarities scores. These similarity values can be also be visualized as similarity matrices such as the one shown in Figure 2.1.

Given a schema matching M, the task is to find all of the matching candidates c which its label value l is `TRUE`.

When considering the networked scenario, a schema matching network $\mathcal{N}$ is a tuple $\mathcal{N} = \langle \mathcal{S}, C, \Omega \rangle$ containing a collection of schemas $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ that should be matched together, the set of matching candidates $C = \{c_1, c_2, ..., c_j\}$, where each candidate is a tuple $c = \langle s, t, l, v \rangle, \forall s \in S_k, \forall t \in S_l, (S_k, S_l) \in \mathcal{S}$. As in the classic schema matching, each candidate c has a vector v of similarity values and a label l associated. The matching network also has a set of network constraints $\Omega = \{\omega_1, \omega_2, ...\}$.

When a schema mathing network task $\mathcal{N}$ is presented, the goal is to find all matching candidates c which its label value l is `TRUE`. The correspondences in the matching network produced should comply with the set of network constraints Ω.

In this work, we consider two network constraints and their modelling will be presented in the following section.

2.2.2 Network Integrity Constraints

When a set of correspondences is generated by automatic methods, they often contain several incoherent matchings creating an *uncertain* matching network. The correspondences that are inconsistent with each other are referred as *uncertain*

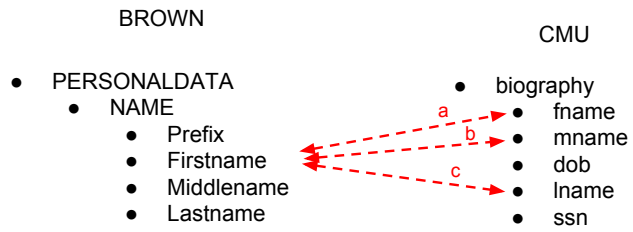

Figure 2.7: One-to-one constraint violation.

correspondences. To help detecting such inconsistencies, some methods rely on the concept of network constraints from studies in mapping messagens between P2P systems [Aberer et al., 2003, Cudré-Mauroux et al., 2006]. These constraints help to stablish natural consistency conditions of a schema matching network. While an inconsistent schema matching network violates at least one integrity constraint, a consistent one has to satisfy all of them [Nguyen, 2014]. The constraints used in this work are domain-independent, other constraints may be added depending on the matching task if needed.

In this work, we consider two types of integrity constraints:

- Type I - *One-to-one constraint*: Any attribute of a schema has at most one matching attribute in another schema. In Figure 2.7, a, b and, c are uncertain matchings as the attribute `Firstname` from Brown has three correspondences in the schema from CMU.

  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`.

- Type II - *Cycle constraint*: Attributes matched together in different schemas must form a cycle. In Figure 2.8, 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`(Brown) and `mname`(CMU) or matching m should be changed to connect `fname`(CMU) and `NAME_MIDDLE`(Cornell).

  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 label $l_z =$`TRUE`.

Notice that a group of matchings that do not violate any constraint can still be incorrect matches, p.e. if we choose $\langle$`Middlename`,`fname`, `NAME_MIDDLE`$\rangle$ in Figure 2.8 as the matching answers, the cycle constraint holds. However, they do not represent the same concept in the real world. Also, a group of uncertain matchings may not have a single correct matching.

Network integrity constraints may not guaranttee that matchings are correct. However, they can be used to find incorrect matches. When a group of uncertain matches is found, at least one of the matchings is guaranteed to be an invalid correspondence. Methods that rely on this constraints have to decide which matching is incorrect by using another source of information such as their similarity values.

Although the network integrity constraints presented can help identify possible mistakes in the network, they do not hold when complex matches occur in the

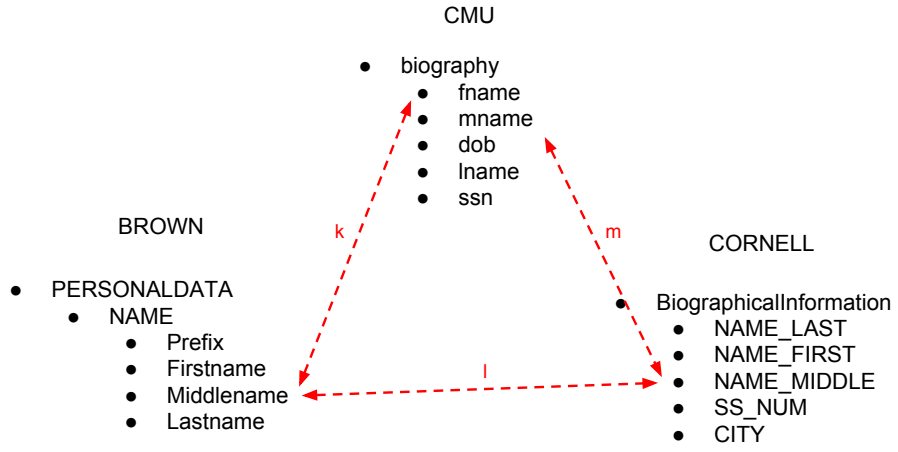

Figure 2.8: Cycle constraint violation.

matching task. For example, if the `BROWN` schema contains an element `FullName` and schema `CMU` contains the elements `FisrtName` and `LastName`, they should be matched together as they represent one concept in the schemas. This problem has been considered in the classic schema matching scenario [de Carvalho et al., 2013]. As other works in schema matching networks [Hung et al., 2014, Nguyen, 2014], we do not consider this type of matchings.

We highlight a couple of proposals that first used the network constraints to help establish the consistency of networks [Aberer et al., 2003, Cudré-Mauroux et al., 2006] and some methods addressing the schema matching problem when a global schema is known and can be matched against [He and Chang, 2003, Madhavan et al., 2005, Su et al., 2006, Nguyen et al., 2011].

The network constraints presented in Section 2.2 were first considered for other areas of study. Aberer et al. [Aberer et al., 2003] address the problem of establishing mappings between P2P systems in order to make them interoperate between themselves. Their method creates a network model between all the peers. Peers start pinging neighbours and spreading messages with their own semantics. Using syntatic similarities as initial scores, these values are propagated to neighbours using the network rules (such as the cycle constraint). In experiments, the authors conclude that with more links between peers, the better is the propagation of the mappings.

The work by Cudré-Maroux et al. [Cudré-Mauroux et al., 2006] also focus on providing a model to enable messages between P2P systems. They consider that there is no *global schema* that every schema can be mapped against. Instead they focus on making local compositions, pair-wise mappings and then propagate them through the network. The method starts by modeling the Peer Data Management Systems as probabilistic graphs. Probabilities between peers are assigned if a mapping is found between such peers. Using the cycle constraint, the probabilities are propagated and with each propagation round, new probabilities are calculated. In the experiments, the authors calculate the accuracy of mappings using different values of a threshold. They found that when using the threshold $\Theta = 0.6$, they achieve at least 0.50 of accuracy. Higher values of threshold do not lead to better results.

Unlike the aforementioned work that consider a global schema does not exist, several works in the schema matching field have been presented making that assumption. They consider that, for each domain, there is a global schema that all schemas in the same domain can be matched against, and, usually, the global

schema is known.

The MGS framework [He and Chang, 2003] aims at matching input schemas by finding correspondences between them and a schema model. The hypothesis is that schemas are being generated by a finite vocabulary of schema attributes, with some probabilistic behavior. The framework has tree steps: *Hypothesis Modeling*, *Hypothesis Generation* and *Hypothesis Selection*. In the step of hypothesis modeling, a synonym discovery approach is used to generate matching candidates and associate them with a probability score. In the hypothesis generation step, consistent models are generated, i.e. sets of matching candidates with non-zero probabilities are selected as long as they do not contradict each other. Finally, in the hypothesis selection step, models are tested with Chi-Square test (χ^2) to quantify how consistent the model is with the data from input schemas. Based on this test, a model is selected as the schema model. The framework was tested with more than 200 sources (web queries) over four domains (movies, music, automobiles and books). In experiments, authors report that the method could find most of the correct matches when there are more schemas to be analyzed. It reached precision of 0.84 and recall of 0.88. We notice the number of matchings in the tasks were small: 8 on average.

Corpus-based Schema Matching [Madhavan et al., 2005] 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 the schemas were defined by different designers, the corpus has a number of representations of each concept in the domain. When new schemas are matched, its elements are matched against the set of concepts. Hence, it can increase the evidence about

a schema element by adding data from the corpus and learn patterns to infer constraints and prune candidate correspondences. The authors use several base learners to match schema elements to concepts in the corpus (such as name and data instance learners). These learners are trained with examples of schema elements that have a known mapping to another element and schema elements that are similar to each other. A meta-learner uses logistic regression to combine the base learners predictions and decide if an element is similar to a concept in the corpus. Once elements are matched to concepts, an augment method is applied to enrich elements information. The augmented model is used to generate several statistics about elements, for instance *neighborhood* and *ordering*. These statistics that are later used to generate and prune candidate matchings. Similar ideas were also used in [He and Chang, 2003]. The experiments were performed with four datasets[5](AUTO, REAL STATE, INVSMALL and INVENTORY). It achieved average precision around 0.83, average recall around 0.79 and average f-measure around 0.79.

Holistic Schema Matching (HSM) [Su et al., 2006] focus on matching across query interfaces. The method uses the network of schemas in a different way: it takes advantage of terms occurrence patterns within a domain to discover matchings. Also, the method can discover complex matchings. It calculates two scores: the *matching score* and the *grouping score*. The matching score reflects the similarity of a candidate based on their cross-copresence count in schemas. This score is calculated as follows: a set of synonym attributes are generated by taking pairs of schema attributes, a heuristic is applied to reduce the number of candidates, each candidate receives a score based on its cross-copresence count, the higher

[5]The datasets are no longer available at the reported web address

its counter, the more likely the attributes are synonym attributes. The grouping score generates higher score between two attributes if they co-occur in the same schema. This score is used to discover complex matchings, for instance, if attributes `first-name` and `last-name` co-occur frequently in schemas, they receive a high grouping score. After calculating scores, a greedy algorithm is applied to generate the list of matchings returned. Candidate pairs with higher matching scores are added to the set of matchings. If a pair contains an attribute already matched, it is added to the set of matchings only if the grouping score is higher than a threshold. Thus, the attributes are grouped composing a complex match. HSM was tested in TEL-8 and BAMM datasets[6], across the two datasets, it achieved average target precision of 0.75 and average target recall of 0.92.

Nguyen et. al [Nguyen et al., 2011] face the problem of integrating several product descriptions from online stores sources into their catalog (their global schema). Each store sells many products from different categories. For each category, the stores have a list of attributes describing products in that category (which can be viewed as a schema). Their task is to find correspondences between the attributes in the stores schemas and their catalog, in order to enable adding product information in the catalog database. The method uses distributional similarity functions and machine learning to perform the matching task.

The work by Alani and Saad [Alani and Saad, 2017] proposes an ontology-based clustering approach to match web-query interfaces (which act as schemas). It has two phases: First, the method creates an ontology by using a semantic knowledge dictionary called XBenchMatch. The elements of the ontology created

[6]Datasets from UIUC Web Integration Repository, available at http://metaquerier.cs.uiuc.edu/repository

are related to each schema attributes. The matching is performed by calculating scores for each attribute based on their labels and values. Scores used include similarity by N-Gram, semantic (using a external dictionary) and TF-IDF. A clustering method based on the work by Williams [Williams, 2010] is used to aggregate similar nodes in the ontology. The authors report results in two datasets reaching F1-score of at least 0.86.

The work by Toan et. al [Toan et al., 2018] addresses the problem by using a probabilistic model to find common concepts in different schemas. Their assumption is that schemas in the same domain are formed by the same concepts, however, concepts might be modelled differently. By using a pre-calculated similarity scores, the method generates a probability for each concept being present in each schema. Then, concepts with higher probabilities classified as being present in the schema. The authors report results in the Purchase Order dataset using different thresholds. The method manages to achieve values of precision and recall from at least 0.6.

2.3 Schema Reconciliation Networks

The matching methods proposed over the years can achieve a good performance on some datasets. However, they cannot be expected to yield a correct result in general. Since matchers rely on heuristic techniques, their result is inherently uncertain. In practice, data integration tasks often include a post-matching phase, in which correspondences are reviewed and validated by an expert [Hung et al., 2014].

The expert can also tune parameters. However, this is the most difficult kind of interaction, as the user must know details on the method and the schemas

involved. In addition, tuning the parameters can lead to trial-and-error use of the method [Do and Rahm, 2002, Lee et al., 2007].

In the case of machine learning based methods, when the user provides matching examples to a method, they usually have to be in the same domain, so the method can learn from them. In this case, the user also should be familiar with the concepts in the domain. This interaction is often performed before the execution of supervised machine learning methods [Doan et al., 2001, Duchateau et al., 2009] and can be performed during the execution of active learning methods.

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 correspondences [Nguyen et al., 2013]. The reconcilation phase often 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 assertion.

As evidenced by Duchateau et al. [Duchateau et al., 2008], 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. Hence, our work addresses the last part of the matching task by taking user assertion on generated matchings, although we briefly address the user participating in the beggining of the matching process as well.

As an example, in Figure 2.9, a matching method generated two matching candidates and presented them to user assertion. The user can accept or reject any of the candidates. A reliable user would accept the matching candidate a - ⟨`Firstname`,`fname`⟩ and reject the candidate b - ⟨`Firstname`,`lname`⟩. Therefore, a

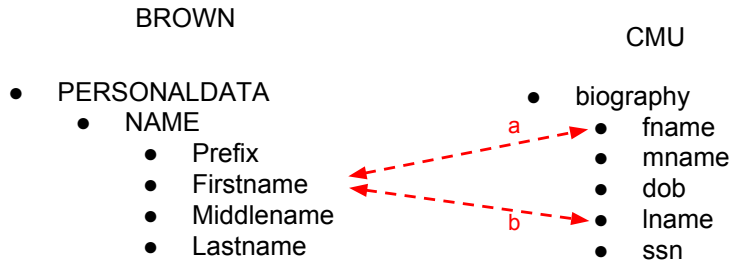

Figure 2.9: User assertion of two matching candidates.

method can correct its matching network with two user assertions.

If a matching method also considers the integrity constraints in its model, the reconciliation phase can be optimized: instead of asking the user to assert the two matching outputs, a method can guide a user to review candidates in a certain order that can minimize her effort. In the same example in Figure 2.9, if an user accepts the matching a - ⟨`Firstname`,`fname`⟩, a method can realize that `Firstname` can no longer have any matching in the `CMU` schema, otherwise it will no longer satisfy the one-to-one constraint. Therefore, the method can automatically reject the matching candidate b - ⟨`Firstname`,`lname`⟩. That way, the reconciliation process queries the user only one time instead of two.

As representatives of methods that adopt the reconciliation approach, we highlight the works of by Hung et. al [Hung et al., 2014] and ArgSM [Nguyen et al., 2013]. While the first presents a probabilistic model to reduce the uncertainty of matching networks, the latter presents a tool to help guiding users in reconciling schema matching networks generated by automatic methods.

The work by Hung et. al [Hung et al., 2014] proposes a model for probabilistic matching networks. The method 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. As the answers given by a matching system are often incorrect, the graph constructed will have violations of the constraints (presented in Section 2.2.2). The method aims at selecting the correspondences with higher confidence score associated and satisfy all integrity constraints. A measure of uncertainty of the network is calculated, when its value is zero it means that there are no violations in the network. The method continues in a loop selecting a correspondence, eliciting user approval or disapproval and updating probabilities of correspondences until the uncertainty reduces to a desired value or it reaches a limit budget of user effort. When the loop stops, all the remaining correspondences are presented as the matched network. The authors did experiments with four datasets (Business Partner, Purchase Order, University Application Form and WebForms)[7] and reported average results of precision (0.85) and recall (0.70). The *a priori* evidence used in this work is the output of semi-automatic matching prototypes. In contrast, other studies rely on matchers similarities.

The next work by Hung et. al [Nguyen et al., 2013] proposes ArgSM, a framework to help the process of reconciling a network of schemas. They propose a representation that captures the experts beliefs and enable reasoning about their inputs. The framework can detect conflicts in assertions and guide the resolution by re-asking users. The collaborative reconciliation works as follows: First, an *individual validation* phase occurs, when each one of the experts evaluate the

[7]Datasets available at http://lsirwww.epfl.ch/schema matching/

same quantity of matches, but different sets (with overlapping matchings). Then an *input combination* phase takes place, when the individual inputs are combined. The correspondences should satisfy all network constraints, if there are conflicting views about a matching, they apply an argumentation algorithm to decide which view to accept.

Others consider using a crowd of users when reconciliating matching networks [Nguyen et al., 2013, Hung et al., 2013]. They consider, besides the assertions of matching answers, the quality of the users based on their domain knowledge and previous assertions. However, we consider these settings out of our scope and do not address this kind of interaction.

To the best of our knowledge, none of the aforementioned pieces of work have a public tool available for testing. To address the participation of the specialist in the reconciliation process experiments, we assume as well as all mentioned works that she will always give correct assertions about matching candidates and we will consider an unlimited assertions 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 a future work.

Chapter 3

Using Machine Learning for Schema Matching and Schema Reconciliation Networks

As discussed in Section 2.1.2, learning methods have been used by several methods in the literature to address the classic schema matching. However, to the best of our knowledge, this kind of technique has not been exploited when considering the networked scenario. In our work, we exploit this approach in a family of methods we present in this chapter. First, we present RF4SM - *Random Forest for Schema Matching*, a machine learning based technique to address the schema matching networks problem. Next, we present RF4SM-B - *Random Forest for Schema Matching Boosting*, which is an extension the first method that acquires training instances with no labelling 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 an-

swers from our method. That way, she can contribute to improve matching quality, but the method spares her from exhaustive work.

3.1 Why a learning approach?

Learning approaches have appeared as the technique of choice for several schema matching methods such those by Li et al. [Li et al., 2005], Madhavan et al. [Madhavan et al., 2005] and YAM [Duchateau et al., 2009]. These methods rely on getting a large set of examples to train models. Once models are created, new schema matching tasks can be provided as input and the method can generate correspondences between the schemas.

Unlike heuristic methods, learning methods can create different matching generators for each new matching task provided. By its deterministic nature, the only way heuristic methods can create new models is if their parameters are changed. However, tuning parameters of such method requires a deep knowledge of the functionality of the method and on the schemas involved. In other words, the designer should know which functions to tune up and which ones to fade down to achieve better matching quality. This instability across different domains can be observed as there is no single method with better matching quality and several new heuristics keep appearing in the literature [Bernstein et al., 2011].

To avoid such directed tunings, we decided to explore using learning approaches. As detailed in Chapter 2, the schema matching task is viewed 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 start the conception of this method by testing several supervised learning approaches to verify if there is a suitable technique to the schema matching networks 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 the ground truth to evaluate such models. We tested as our base learning strategies: the Decision Tree (J48) [Quinlan, 1993], AdaBoost algorithm [Freund and Schapire, 1997], Logistic Regression [Walker and Duncan, 1967], Random Trees [Ho, 1995] and Random Forest [Bishop, 2006]. All of the learning strategies available in the Weka package [Hall et al., 2009].

Figure 3.1 illustrates how we carried out the testing process. For each matching task, a random set of instances (matching examples) was taken as the *training examples* I_{tr}. Next, the same set of examples is given to each classifier that learns a different supervised model (Sup_Model) according to its strategy. Finally, models are tested with the remainder of unlabeled instances. Each model generates a label for each matching instance: if elements correspond to each other, the label given is `TRUE`. Otherwise, the label given is `FALSE`.

Formally, given a labeled training set of examples I_{tr}, perform training according to a supervised learning algorithm and create a model Sup_Model that will classify the remainder of unlabeled instances, the test set I_{ts}. In Chapter 4, we detail this experiment and the results obtained for each of the learning strategies tested. The best supervised learning algorithm according to that experiment was chosen to be the base classifier of the method. This draft method is described in Algorithm 1.

The draft method works by taking a random subset of examples and providing

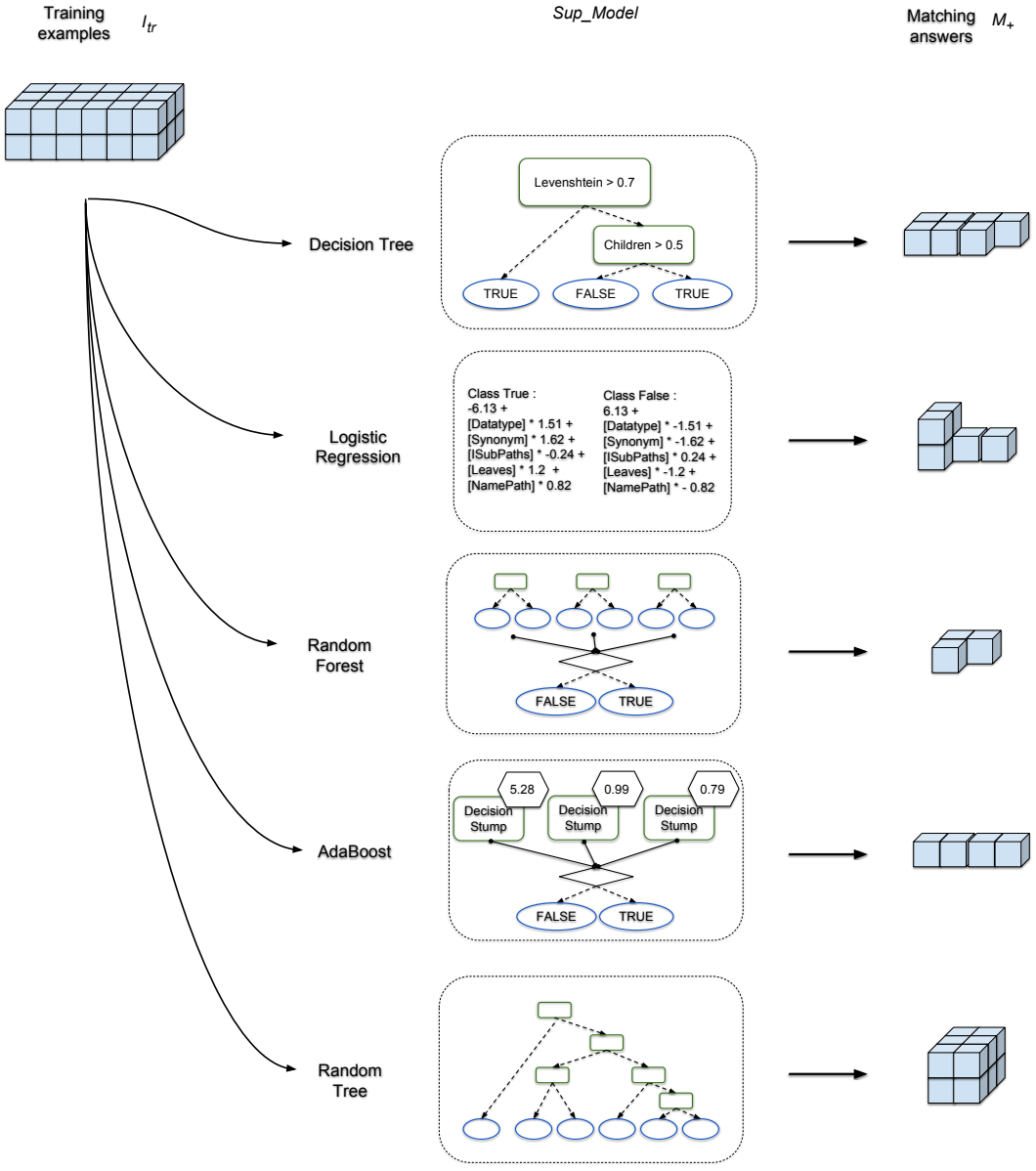

Figure 3.1: Draft Algorithm flow.

Algorithm 1 Draft_Algorithm(C, p)

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

them as training for a supervised learning algorithm. Once the learning occurs, a model is generated and it can be used to classify the remainder of the instances. All the matches found by the model compose the matching network.

3.2 Choosing a Learning Approach - RF4SM

The experiments for choosing the most suitable approach to the schema matching networks problem are further detailed in Section 4.2. Our conclusion is that *Random Forest* was the most suitable algorithm and it was chosen as our base classifier. The Random Forest algorithm is simple and flexible enough to be applied to different types of input [Bishop, 2006] and easy to implement.

Besides better matching quality, other methods (such as Yet Another Matcher [Duchateau et al., 2009, Ngo and Bellahsene, 2012], the work of Rong et. al [Rong et al., 2012], ALMa [Rodrigues et al., 2015], and the work of Reis et. al [Reis et al., 2017]) used forests of trees or random forests as learning algorithms to address the matching problem. This fact also backed up our decision to go with decision trees as the basic learning approach.

Once we decided on the base classifier, the supervised learning method for

schema matching networks can be defined according to Algorithm 2. We modify the *Draft Algorithm* by setting the Random Forest as the default learning algorithm. From this point forward, the strategy will be referenced as `RF4SM` - *Random Forest for Schema Matching* and it is depicted in Figure 3.2.

`RF4SM` receives as input a network of schemas to be matched. By combining all possible pairs of elements between schemas, a set C of matching candidates is created. 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 which get the label `TRUE` will compose the answer of the method, i.e. the schema matching network M_+.

Algorithm 2 RF4SM_Algorithm(C)

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

This simple training and testing technique has been used to address the classic schema matching problem. One can try create a different model for every pair of schemas in a matching network and unify their matches. The issue with this approach is that is hard to learn good models with a small amount of examples. Also if the matching tasks are addressed separately, no common knowledge can be transferred from one to another as opposed to the networked scenario where all data from schemas are in one task.

`RF4SM` takes a labeled set of examples as training and generates a learning

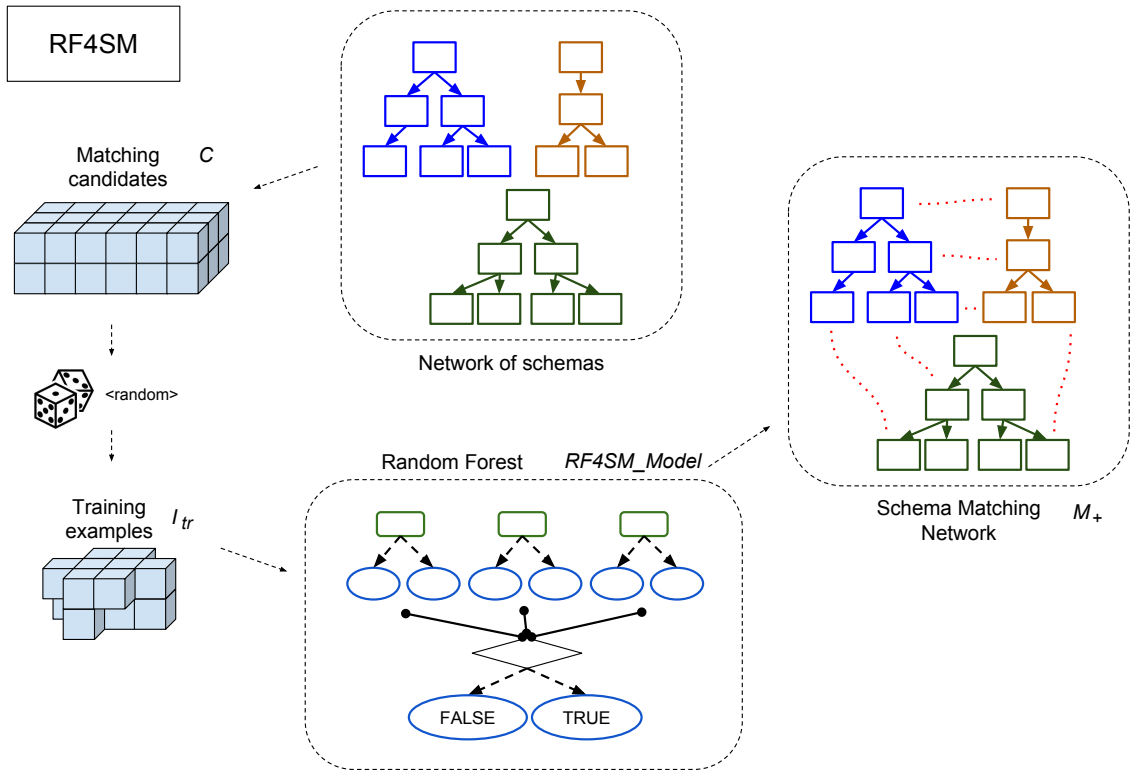

Figure 3.2: RF4SM algorithm flow.

model. The generated model then evaluates the matching candidates from the network and classifies them as `TRUE` if the candidate is a valid correspondence or `FALSE` otherwise. In cases where a large set of labeled examples is obtained without asking the user for labels, `RF4SM` can be a suitable option, such as when a crowd of users is available for labeling. However, that might not always be the case. Also, the schema matching networks problem requires that *network constraints* are obeyed.

To cope with that scenario, we evolved our method to take in consideration both facts and make it more suitable to the problem. The next goal is to generate a network of matchings with no inconsistencies while sparing the user the effort of

labelling instances.

3.3 RF4SM-Boosting

A supervised strategy can achieve good results regarding the matching quality. However, it requires a massive training set of labeled examples making it unfeasible in real applications as the examples must be revised by an human expert.

As a successful approach in the classic schema matching, the supervised method faces new challenges when considering the networked scenario: the amount of data in the task is much bigger, the amount of training required might be unfeasible or costly, and there are network constraints that should not be violated.

3.3.1 *Unfiltered*-RF4SM-B

To overcome the scalability issue, we decided to test an automatic training approach. We depict this strategy in Figure 3.3. We are going to boost the learning process by taking the answers of a base method, more specifically, a heuristic method generates a set of positive matchings (A_+) which automatically label the matching candidates used in traning. Similar to `RF4SM` (presented in Section 3.2), the combination pairs of all schema elements form the set of matching candidates C.

Those candidates which were evaluated as positives by the base method receive the label `TRUE`. By default in heuristic methods, all the remaining matching candidates receive the label `FALSE` (A_-). As the number negative examples are much bigger than the positives, only a random subset of this set is used in training. Once the training set is automatically created, it is submitted to the learning algorithm

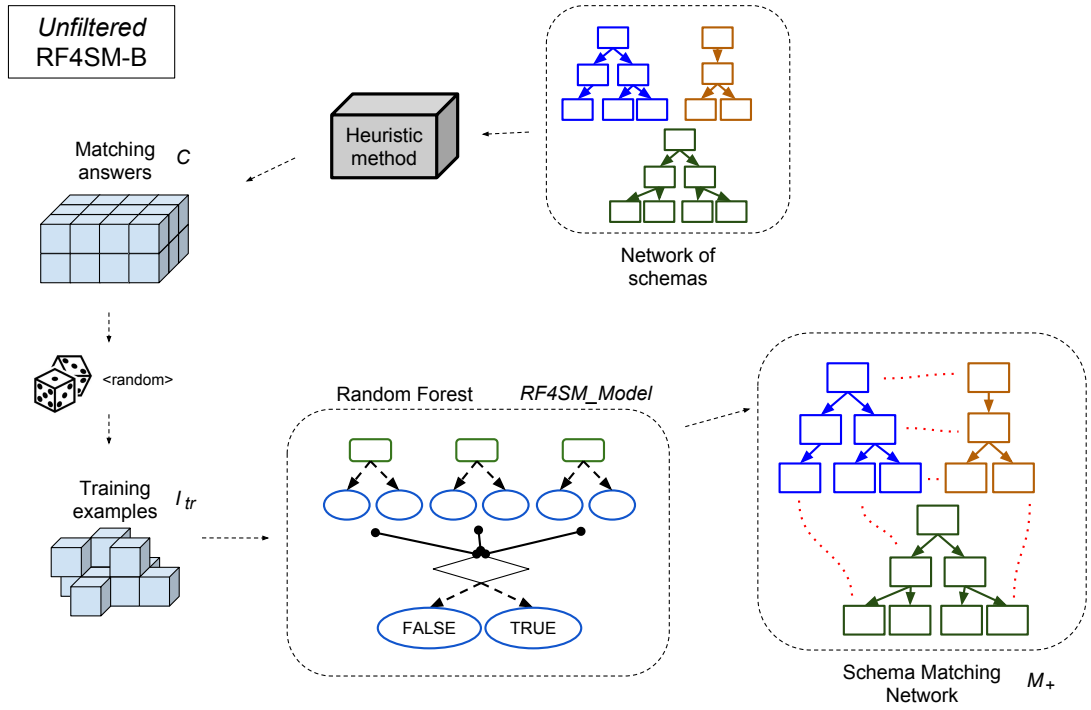

Figure 3.3: Unfiltered RF4SM-B algorithm flow.

based on Random Forest and the `RF4SM-B` model is generated. Finally, the model is used to classify all matching candidates and those evaluated as `TRUE` matchings will compose the schema matching network.

Heuristic methods often generate incorrect matching answers. Hence, the automatically labeled training might contain many mislabels. As no prunning strategy is applied to remove possible mislabels, this method is referenced as `Unfiltered RF4SM-B`. This strategy is described in Algorithm 3.

This approach is able to gather a great amount of training examples. As evidenced by further experiments (see Section 4.3.1), this process can address the issue of amount of examples. However, as automatic methods do not consider the network constraints, they often generate inconsistent matchings containing

Algorithm 3 *Unfiltered*_RF4SM-B_Algorithm(C, A_+)

C is the set of all maching candidates from the network
A_+ is the set of positive answers given by the base method
1: $A_- \leftarrow I \setminus A_+$
▷ by default, all non-positive answers from a base method are considered as negative
2: $A_- \leftarrow randomSubset(A_-, k)$
▷ k is the size of the random negative subset of instances
3: $I_{tr} \leftarrow A_+ \cup A_-$ ▷ the training set is the union of positive and negative sets
4: $RF\_model \leftarrow randomForestSupervisedLearning(I_{tr})$
▷ training procedure
5: $(M_+, M_-) \leftarrow classify(RF\_model, I)$ ▷ classifying instances
6: **return** M_+ ▷ matching candidates classified as `TRUE`

possible false positives, making the automatic training lack in quality of the examples. When these mislabeled matchings are used as training, they can confuse the learning process creating even more inconsistent matching networks.

3.3.2 *Filtered*-RF4SM-B

In order to obtain a *cleaner* automatic training set, we also consider taking the set of positive answers given by the base method and submitting them to the network constraints. The rationale is to avoid false positives in the training set and possibly learning better models. Also, the network constraints help to identify negative examples, which in `Unfiltered RF4SM-B` were taken as the default answer of the heuristic method.

For example, in Figure 3.4 (which we repeat in this chapter, for better reading), 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. Using this rationale, we can automatically generate `FALSE`

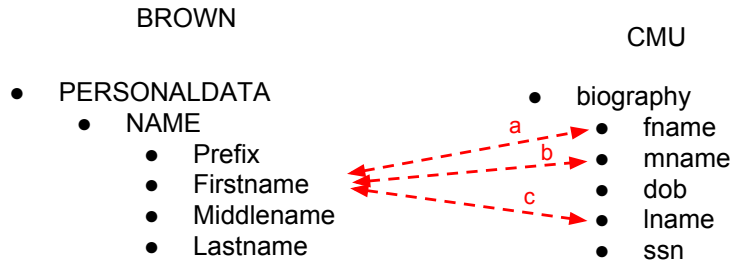

Figure 3.4: One-to-one constraint violation.

matchings to our training set. Hence, we stop taking negative examples from the heuristic method.

In Figure 3.5 we depict the flow of the `Filtered-RF4SM-B` algorithm. First, a set of schemas is submitted to the (heuristic) base method. It generates a set of positive matches A_+. Unlike *Unfiltered*-RF4SM-B (presented in Section 3.3.1), the set A_+ is filtered using the network constraints (previously presented in Section 2.2.2). 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 used the network constraints (as mentioned in the example above). 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 `Filtered RF4SM-B` model is generated using the Random Forest algorithm. Finally, the model is used to evaluate the matching candidates. Candidates classified

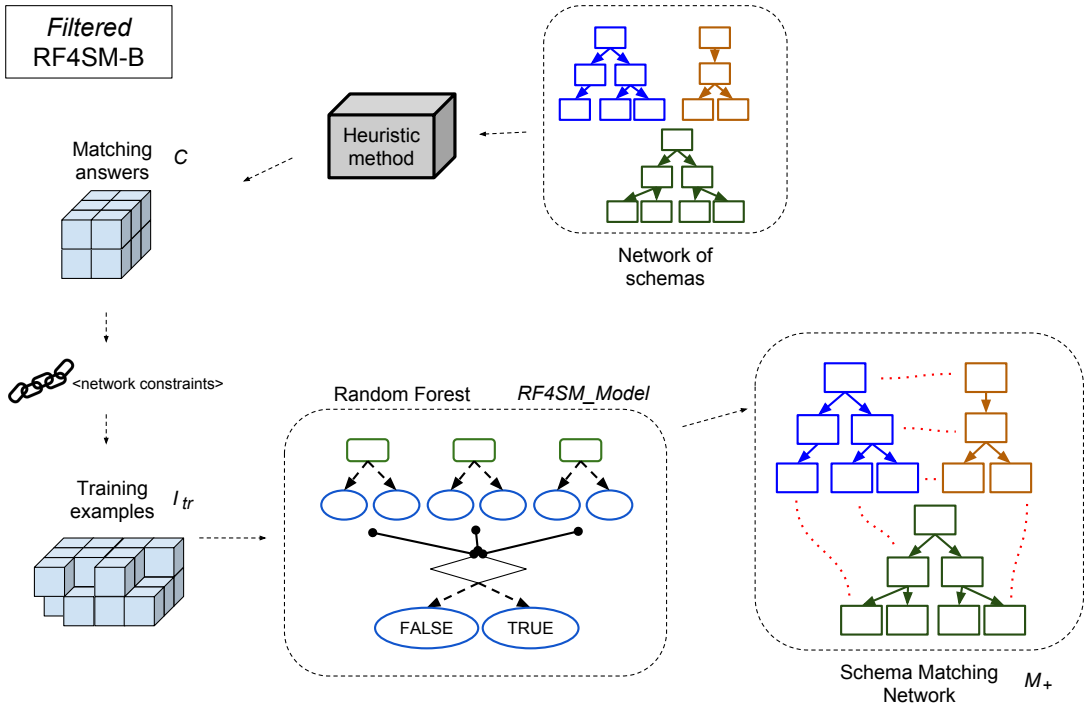

Figure 3.5: Filtered-RF4SM-B algorithm flow.

as `TRUE` matchings compose the schema matching network.

As in `Unfiltered RF4SM-B` algorithm, all of the positive matchings found (M_+) are used to build the schema matching network. In the further experiments in Section 4.3, we test both strategies to verify how the filtering process changes the models learned and the impact caused when evaluating the matching answers generated. We verify that besides a smaller training set, the filtered set has better quality and it leads to better matching quality.

The `Filtered RF4SM-B` method is an evolution of `RF4SM-B` that eliminates the user participation in labeling examples. A further improvement would be to use the user knowledge in the *reconciliation matching network* task, which is, usually, less laborious than labeling training examples. Also, at this point, a network of

Algorithm 4 *Filtered*_RF4SM-B_Algorithm(C, A_+)

C is the set of all maching candidates from the network
A_+ is the set of positive answers given by the base method
1: $(A_+, A_-, U) \leftarrow networkRestrictionsFilter(A_+)$
▷ the network restrictions produce three sets: the accepted instances A_+, the rejected instances A_- and the uncertain instances U
▷ A_+ is the set of matchings that were given as `TRUE` and accepted by the network restrictions
▷ A_- is the set of matchings that were rejected by network restrictions. Recall Figure 3.4, when one matching is accepted as `TRUE`, other matchings can be taken as `FALSE`
▷ U is the set of uncertain matchings, i.e. they conflict with each other.
2: $I_{tr} \leftarrow A_+ \cup A_-$ ▷ the training set is the union of positive and negative sets
3: $RF\_model \leftarrow randomForestSupervisedLearning(I_{tr})$
▷ training procedure
4: $(M_+, M_-) \leftarrow classify(RF\_model, I)$ ▷ classifying instances
5: **return** M_+ ▷ matching candidates classified as `TRUE`

matchings is already built, which means this step is optional and only needed if a boost in the quality of the matchings is required.

3.4 *Filtered*-RF4SM-B-Reconciliation

As it occurs after the schema matching process finishes, the reconciliation task can be optional. However, it is an important step as it can help to enhance 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 (compared to tuning the parameters of a method) apart from knowing the schemas involved. Also, it is smaller than the job of labeling training examples. Moreover, the networked scenario magnifies the task size and the task requires an eased user effort to avoid

human errors and lower the cost of the process.

Some methods [Hung et al., 2013, Nguyen et al., 2013] consider that a crowd of users is available. However, we do not consider this option as other factors have to be considered such as reliability of a user. We consider that only one user is available and she is reliable to perform the reconciliation. Our goal is to verify how much the quality of matching results can be improved. We leave the optimizations of that task for future researches.

The reconciliation process is described in Figure 3.6. After the schema matching network task is performed by *Filtered*-`RF4SM-B`, a set of matching answers M_+ is generated. This set of answers is submitted to the network constraints in order to obtain a consistent set of answers. The uncertain matchings U_+ are given to the user for reconciliation, the user can either accept an uncertain matching labeling it as a `TRUE` matching or rejecting it (hence labeling it as `FALSE`). The final matching answers are the union of both sets of classified `TRUE` matchings by the method and the accepted matchings by the user. The `RF4SM-B-Rec` algorithm is described in Algorithm 5.

The user effort by only reconciliating the uncertain matches is reduced (compared to evaluating all of the answers). Further optimizations can be performed to reduce the quantity of user assertions, such as changing the order uncertain matches are presented to the user [Hung et al., 2014] and using a crowd of experts to evaluate answers, we leave that to future work. Our intent in this study is to verify how a user can contribute to the schema matching network task while keeping the reconciliation a feasible task.

Section 4.4 presents the experiments performed to verify the improvement provided by a user when reconciliating the results of `RF4SM-B`. We compare the effort

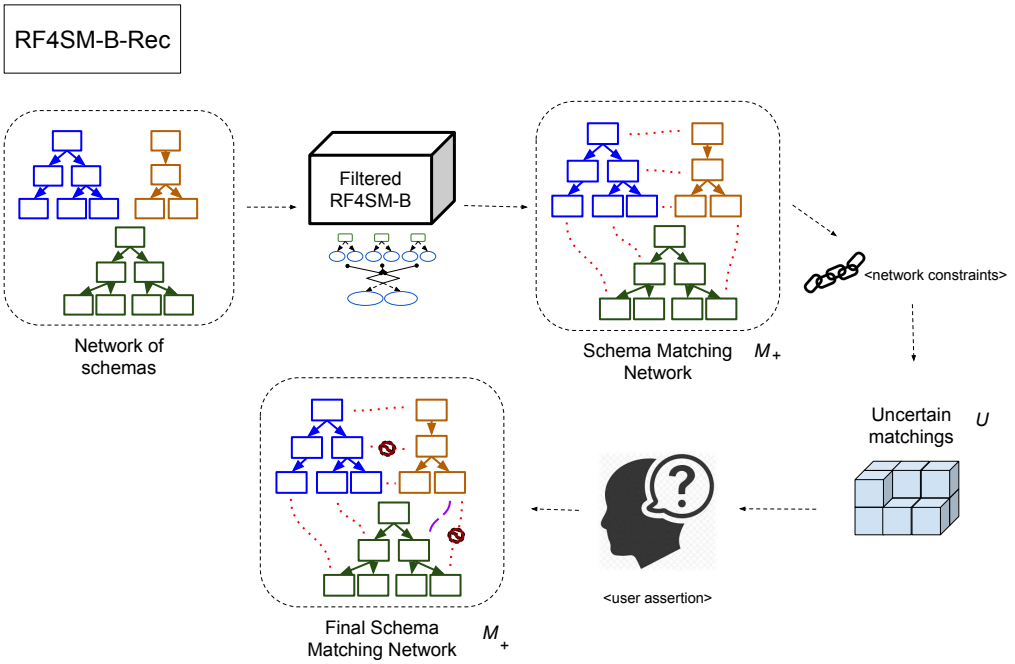

Figure 3.6: *Filtered*-RF4SM-B-Rec algorithm flow.

made by a user when reconciliating automatic methods and `Filtered-RF4SM-B` and how the use of different base systems impact in the final stage.

As evidenced by our experiments, we conclude that it is more advantageous to first prune uncertain matching candidates before giving them to user assertion. We observe that their work can be highly reduced up to 6 times while achieving similar levels of final matching quality.

3.5 Summing up

In this chapter we presented a family of methods to address the Schema Matching Networks problem by using machine learning techniques. First, in Section 3.2, we presented Random Forest for Schema Matching - `RF4SM`. This first method

Algorithm 5 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: $M_+ \leftarrow Filtered\_RF4SM\text{-}B\_Algorithm(C, A_+)$
▷ Filtered-RF4SM algorithm presented in Algorithm 4
2: $(M_+, M_-, U) \leftarrow networkRestrictionsFilter(M_+)$
▷ the network restrictions produce three sets: the accepted instances M_+, the rejected instances M_- and the uncertain instances U
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`

leverages the use of a machine learning algorithm to address the schema matching networks problem. Machine learning approaches have the advantage over heuristic approaches as they can yield specific models for different matching tasks, whilst heuristic approaches heavily rely on an expert for tunning the method and finding the best heuristic for every single task.

Next, we presented `RF4SM-B` in Section 3.3, a method that uses a boosting strategy to automatically acquire training examples. Hence it eliminates the user participation in labeling examples. The training examples are acquired by taking answers from a heuristic approach. In `RF4SM-B` this set of answers were handled in two ways: by applying or not a filter based on *network constraints*. *Unfiltered*-`RF4SM-B` does not uses the filter when taking the automatic training. On the other end, *Filtered*-`RF4SM-B` uses network constraints to prune inconsistent matchings from the training. The cleaner the training set, the better the models are learned leading to better matching quality in terms of precision.

As the user was spared of labeling examples in the begginning of the matching process, she can now be used to assert the schema matching network generated by

our method. This is the process knows as the *schema reconciliation*. In Section 3.4, we present `RF4SM-B-Rec`, a method that uses the network constraints and user assertions to enhance the matching quality. As the user provides feedback, her assertions can be used to remove incorrect matchings. Also, using the network constraints, the method can derive undiscovered matchings and prevent the user of asserting unnecessary mismatches.

In the following chapter, we present experiments that corroborate our hypothesis that cleaner training sets yield to better models, evaluate how much "dirty" training there is in the training sets and its impact in the matching networks. We also present experiments verifying that our method yield less amount of user effort in the schema reconciliation process.

Chapter 4

Experiments

In this chapter, we present an experimental evaluation of the methods we proposed and presented in the previous chapters. Our goal is to validate our assumptions and claims, by evaluating how our methods perform running over datasets previously used in schema matching experiments.

4.1 General Settings

Datasets. All of the experiments presented in this study will feature a collection of five datasets. Each dataset represents a task of matching a network of schemas, and contains more than two schemas. All of the schemas in a dataset belong to the same domain. An overview of the datasets characteristics is presented in Table 4.1. As indicated in this table, all these datasets were already used in previous work.

There are five domains with a certain number of schemas that compose a network. Also, each pair of schemas in each network has different numbers of matches between them. We stress that the actual number of matches to be found

Table 4.1: Datasets characteristics. For better reading, we present the references cited in this table here. A-[Duchateau and Bellahsene, 2010], B-[Aumueller et al., 2005], C-[Drumm et al., 2007], D-[Hung et al., 2014], E-[Su et al., 2006], F-[He and Chang, 2003], G-[Do and Rahm, 2002], H-[Rodrigues et al., 2015].

Domain	**Betting**	**Business Partner**	**Magazine Subscription**	**Online Book**	**Purchase Order**
#Schemas in 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
Used in	A, B	C, D	E	E, F	D, G, H

is actually very low compared to the number of candidate matches, making it a challenging task. Even though these datasets have been used in previous work, some of the datasets were made available without their ground truths. Therefore, we had manually matched the schema networks in advance performing experiments with the methods. For this study, complex matches were not considered and were discarded. Also, the ground truth of the matching networks generated are guaranteed to obey all of the network constraints.

Hardware. All the experiments presented in this study were performed in a computer with the following configuration: 64-Bits-Ubuntu 14.04.1 LTS operational system, QuadCore Intel Xeon (2.9 GHz) processor and 9995 MiB memory.

Evaluation Metrics. Let M be the set of all matching candidates, C be the set

of correct matchings of a matching network and $C \subset M$. 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* ($\frac{|A \cap C|}{|A|}$), *recall* ($\frac{|A \cap C|}{|C|}$) and *F1-score* ($\frac{2 * precision * recall}{precision + recall}$). Notice that, due to unbalancing of valid and invalid matching candidates, the invalid candidates being the large majority, we only consider the positive candidates in our evaluation. Also, finding the valid matching candidates is the real goal of a matching task.

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

To compact both metrics in one value, often the F1-score is used. It is a harmonic mean of the two values. It treats both precision and recall as equally important scores. Duchateau et. al [Duchateau et al., 2008] argues that, in a user point-of-view, recall is more important than precision, as it is harder to find undiscovered correpondences than to assert incorrect correspondences returned by a method. In this work, we do not intend to propose a new evaluation method. Hence, we stick with F1-score to represent an overall quality measure of matching results.

Base Systems

The `RF4SM-B` and `RF4SM-B-Rec` methods will use a black box system as its base provider of matching candidates. We used as the base systems unsuper-

vised methods that use heuristic approaches, namely, the well-known methods COMA/COMA++ [Do and Rahm, 2002, Aumueller et al., 2005] and Similarity Flooding [Melnik et al., 2002]. Both base systems are available online [1][2].

We selected the library of matchers available in COMA/COMA++ and Similarity Flooding to be the base matchers for all learning techniques presented. 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 aforementioned metrics.

In COMA/COMA++, we used the best combination of matchers and parameters reported by the authors. They report that in its best configuration, COMA runs all the hybrid matchers, agregates matrices by *Average*, matches in *Both* directions and selects matches by 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* between schemas elements names, then applies the *flooding algorithm* that propagates the similarities through nodes in the graph. Finally, it applies filters to select the matchings.

All of the results reported for machine learning algorithms are the average of 30 runs, each one with a different random seed.

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

Baselines

As representatives of heuristic approaches, we will also use COMA and Similarity Flooding as baselines. To the best of our knowledge, both methods were never presented addressing the schema matching networks 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 proposed in the literature for the classic schema matching scenario [Duchateau et al., 2009, Gal and Sagi, 2010, Nguyen et al., 2011, Rodrigues et al., 2015]. Thus, instead of adapting each method to the networked scenario, we constructed network schema matching methods with their core classification algorithms, and run experiments with them.

The methods for the classic schema matching use all kinds of matchine learning algorithms: the work of Gal [Gal and Sagi, 2010] uses the statistical-based method *Naive-Bayes*; YAM [Duchateau et al., 2009] uses tree-based algorithms such as *J48*, functions such as *Logistic Regression* and *Bayes Networks*; the method by Nguyen et. al [Nguyen et al., 2011] also uses *Regression* and ALMa [Rodrigues et al., 2015] also is tree-based.

4.2 Evaluating Machine Learning Algorithms for the Schema Matching Networks Problem

In this section we present two experiments. The first was designed to evaluate different machine learning methods previously used for the classic schema match-

ing scenario (Section 4.1) in the context of the schema matchig networks 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 [Hall et al., 2009]. We ran each classifier with five different sizes of training sets, ranging from 10% to 50%. The remainder of the instances were used in the test set. All the results reported are the average of 30 runs. All the algorithms were tested in the five previously presented datasets. For the first experiment, the average of the results in the five datasets are shown as the final result.

For details of the algorithms parameters, please refer to the Weka documentation[3]. The specific settings for each algorithm were set as follows:

- **AdaBoost:** $weightThreshold = 100$, $baseClassifier =$ DecisionStump, $numberOfIterations = 10$;
- **J48:** $unpruned = false$, $confidenceFactor = .25$, $subtreeRaising = true$, $binarySplits = false$;
- **Logistic Regression:** $maxIts = -1$, $ridge = 1.0E - 8$;
- **Random Forest:** $numTrees = 100$, $maxDepth = unlimited$;

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

- **Random Tree:** $KValue = random$, $maxDepth = unlimited$, $minNum = 1.0$

- **SVM:** $svm\_type = 0$, $kernel\_type = 2(radial)$, $degree = 3$, $gamma = 1/num\_features$, $cost = 1$, $nu = .5$, $epsilon = 0.1$, $epsilon(tolerance) = 0.001$, $weight = 1$.

4.2.1 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. To each of the five datasets, we randomly divided the instances in two groups: training and test sets. The training set is given to each of the classifiers to build models, then the test set is submitted to the models and we evaluated the classification according to aforementioned 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 Figure 4.1. The Random Forest classifier achieved the highest average of F1-scores consistently when increasing the training set. We point out that classifiers based on ensemble of trees were also chosen as the base classifier in other works such as YAM [Duchateau et al., 2009] and ALMa [Rodrigues et al., 2015]. With this experiment, we established that Random Forest were the base classifier for the method. From this point, it will be referenced 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 [Duchateau et al., 2009,

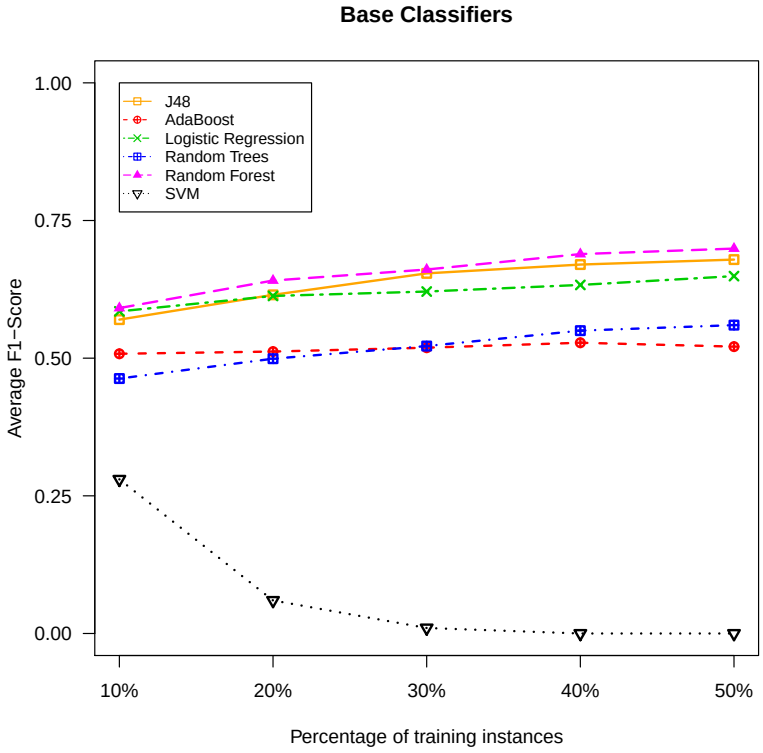

Figure 4.1: Averaged F1 scores reached by each classifier with different sizes of training.

Gal and Sagi, 2010, Nguyen et al., 2011]. With that in mind, we decided to take the learning algorithms as the representatives of these methods and considering them as the baselines of this work for the schema matching networks task.

4.2.2 RF4SM x Heuristic Strategy - 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 experimental dataset. Our goal with this experiment was to find the amount of learning instances required to achieve results comparable to the base methods. This amount ultimately corresponds to the effort needed by a user for labeling examples before the matching process in a supervised setting.

For brevity, we opted to show only the precision, recall and F1-scores achieved by `RF4SM` using 30%, 40% and 50% of training, alongside with the base methods in Figure 4.2. 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).

The downside to this method is that it needs a massive amount of training examples. Ultimately, the labeling of instances translates into user-effort. While `RF4SM` makes use of the user effort, `COMA` and `SF` are heuristics, therefore they might seem to be the most advantageous choice in a real application. However, heuristic methods rely heavily on tuning of parameters and, depending on the heuristic used, they only can be applied to a finite set of application domains.

In absolute numbers, `RF4SM` needed an average of 16800 examples to train its

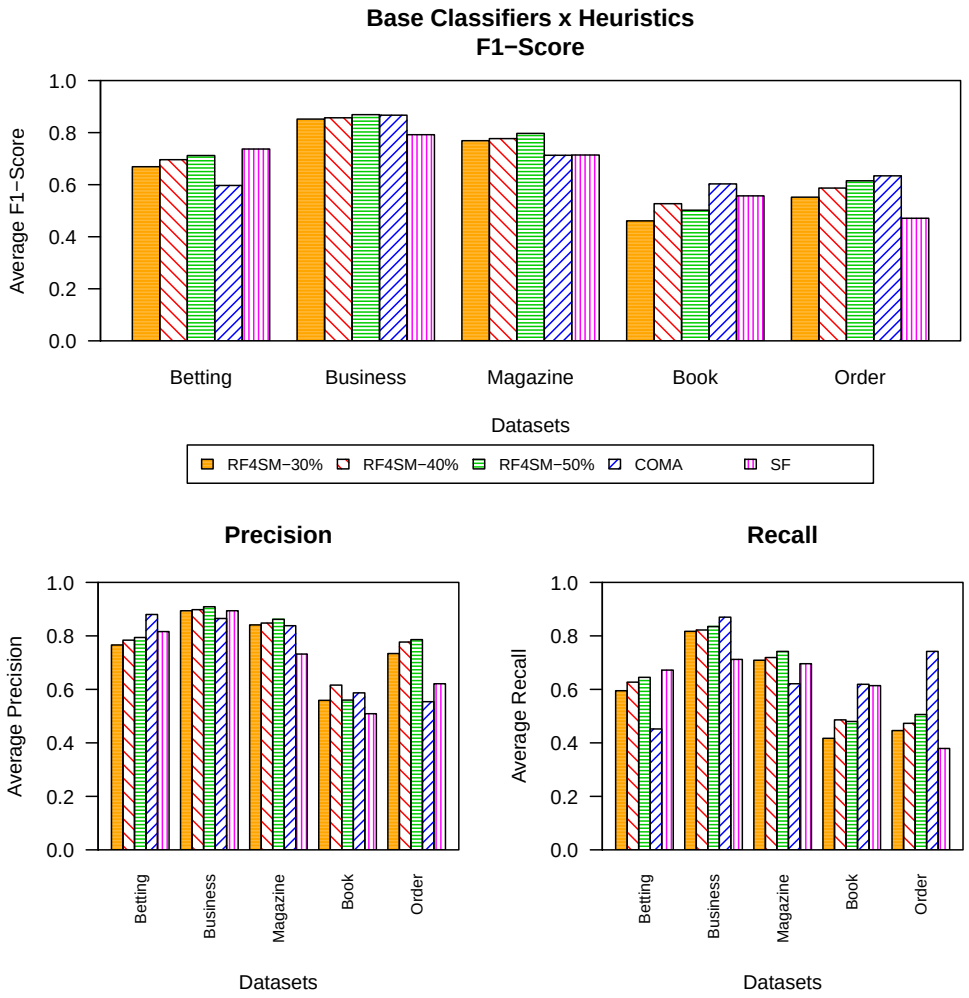

Figure 4.2: Average results of supervised and heuristic methods in five datasets.

models, what is a high number of instances to get with a high cost as they must be labeled. This issue will be addressed in the next set of experiments.

4.3 RF4SM-B: Experimental Evaluation

The following experiments were designed to test our boosting strategy that uses an automatic labeled training set. `RF4SM-B` was tested using two different training

sets, the first is composed by the answers produced by the base systems (*Unfiltered-RF4SM-B*) and the latter is composed by filtered answers by network constraints (*Filtered-RF4SM-B*). Additionally, we performed an experiment to verify differences in the training sets such as accuracy and quantity of positive examples.

Setup. In this set of experiments the methods were evaluated in all five datasets (presented in Table 4.1). 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 (`COMA` and `SF`) and the models were trained with the two aforementioned training sets. Base methods featured in this section were run with their default settings, as described in Section 4.1. Both COMA and Similarity Flooding are used as base methods and they represent the heuristic methods baselines for this set of experiments. In results reports, *Unfiltered-RF4SM-B* will appear as *Unf-RF4SM-B* and *Filtered-RF4SM-B* will appear as *Filt-RF4SM-B*.

4.3.1 Training with Automatically Labeled Examples

As described in Chapter 3.3, aiming at reducing the user effort to zero, we performed the next experiment by taking the answers from the base methods and used them as training set to `RF4SM`. We also used the network constraints to create a filter and created a second training set of examples. Both sets were submitted to the learning process and the models generated were evaluated in the five datasets previously presented.

We show the average precision, recall and F1-scores of *Unfiltered*-`RF4SM-B` when using answers from `COMA` in Figure 4.3 and `SF` in Figure 4.4.

We notice that *Unfiltered*-`RF4SM-B` struggled to keep the same levels of precision

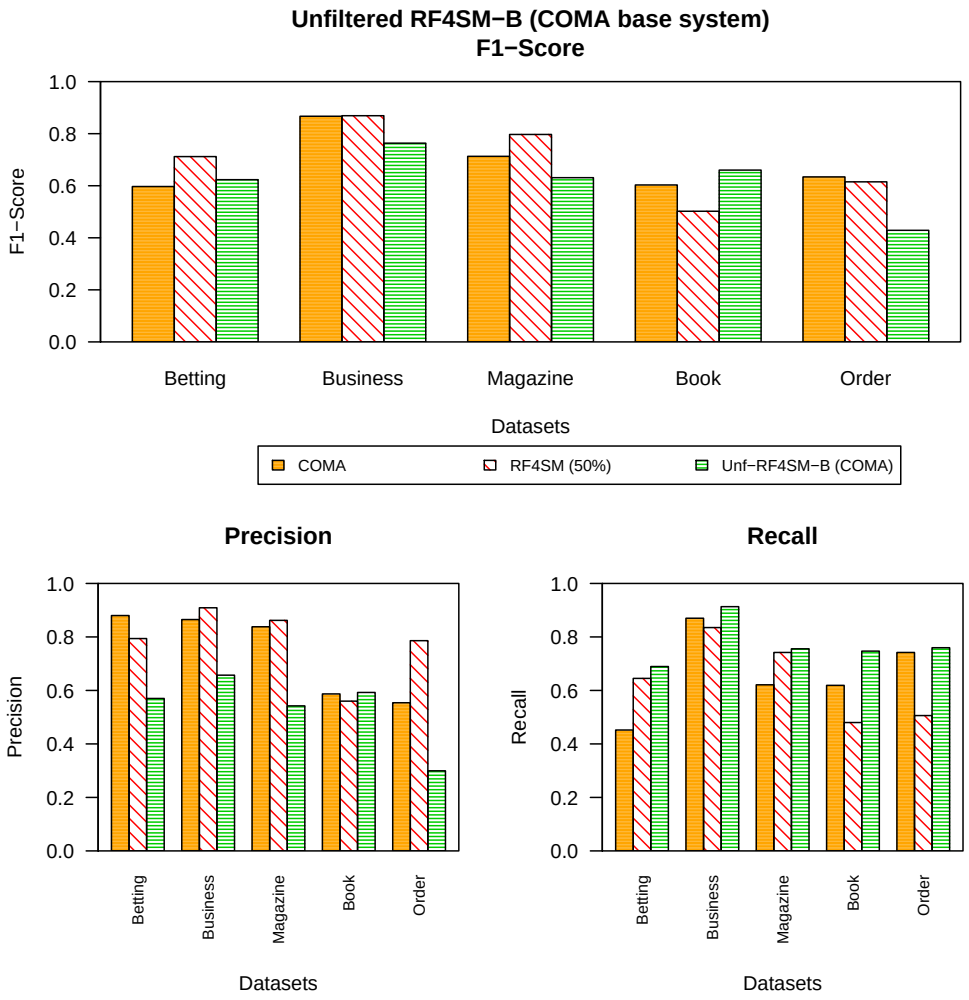

Figure 4.3: Average results obtained by `RF4SM-B` using as training set the answers taken from `COMA`.

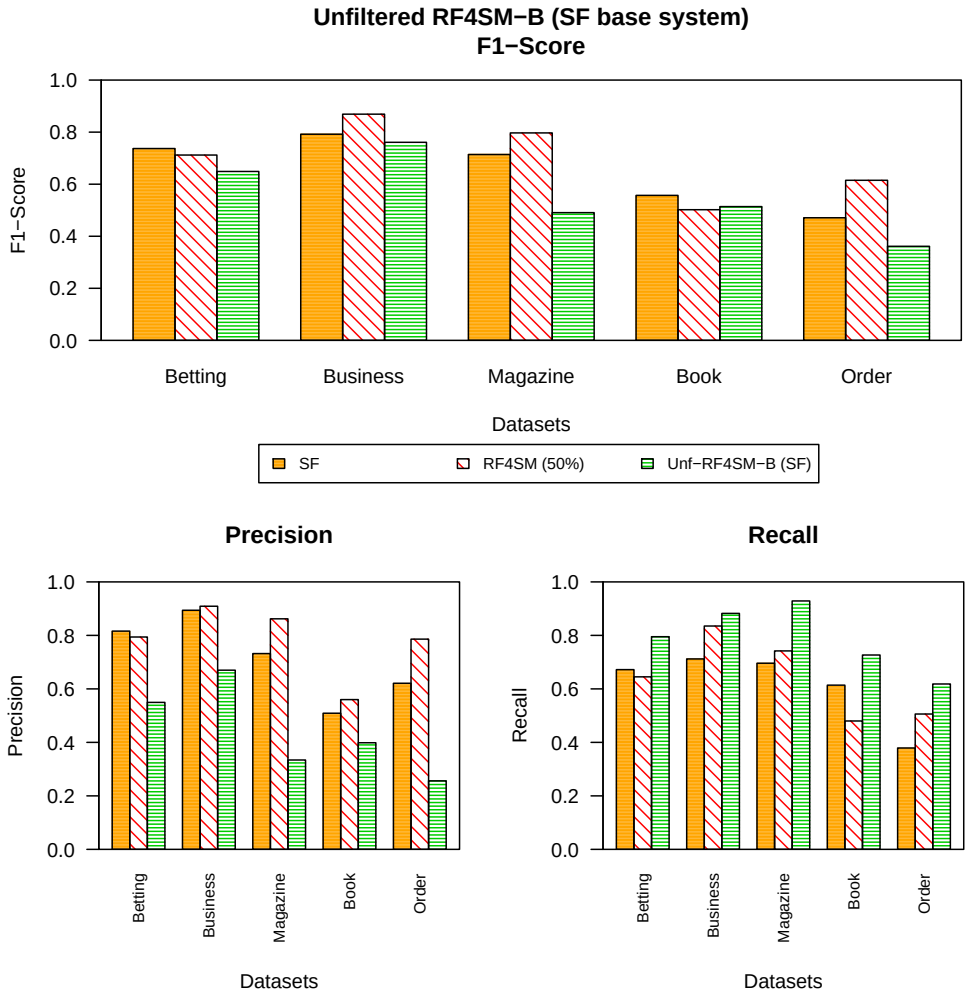

Figure 4.4: Average results obtained by `RF4SM-B` using as training set the answers from `Similarity Flooding`.

as the supervised strategy RF4SM. Generally, the worst the precision of the base method, the worst the boosted learning. On the other side, the 'not so correct' training of *Unfiltered*-RF4SM-B allows it to discover a lot more false negatives than the other methods, raising recall by using both base systems.

On average, almost 80% of the matchings were found by *Unfiltered*-RF4SM-B. This high recall value might be important if a user is available to perform a post-matching task, where she can look the answers of the method and remove false positives, automatically improving the precision of the matching network generated. Regarding F1-score, on average COMA and RF4SM (40%) tie at 0.68, RF4SM (50%) reaches 0.70 and *Unfiltered*-RF4SM-B reaches 0.63.

The low values of precision indicates that the models generated might be incorrect, or at least, too general, thereby generating a lot more matchings than it should. On that thought, we decided to investigate the quality of the training set generated for RF4SM-B in the next experiments.

4.3.2 Leveraging Network Inconsistencies

The network constraints presented in Section 2.2 are good indicators of the integrity of the matching network. As such, they can be used as a "tool" to identify possible incorrect matches and also to find possible missing correspondences. Therefore, when the network has a high number of inconsistencies (i.e., the network is uncertain), it is an indicative 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.

Table 4.2: 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

Table 4.3: Inconsistencies types

	Constraint associated
Type I	one-to-one constraint
Type II	cycle constraint

To build the networks of answers, for each of the five domains we submitted every combination of pairs of schemas to the unsupervised methods. After that, all of the methods answers were gathered and inserted in 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 (see Section 2.2.2). All the results are shown in Table 4.2. For convenience, Table 4.3 summarizes the types of inconsistencies associated with the presented in Section 2.2.2.

We notice that methods tend to generate more inconsistencies when there are more schemas to be matched. Also, we stress that the Similarity Flooding network of answers did not violate the constraint of Type I (*one-to-one constraint*) not even once. This might be caused by its more "graph-oriented" approach.

4.3.3 Training Set Quality

An inconsistent network of answers can be an indicative of mislabeled matchings. In order to test the quality of the training set provided, we designed an experiment

to verify if the automatic training sets provided contains useful instances (positive examples - as they are more rare) and accurate labels in some degree that do not misleads the learning process. As observed in Section 4.3.1, inaccurate labels in training sets lead to a confusing process of learning, generating poor models.

As show in Table 4.2, the answers of automatic methods are quite inconsistent making them less reliable. In the next experiment, we took the answers of both `COMA` and `SF` and applied to them the network constraints, the inconsistent matches were removed from the set of answers.

We notice the number of positive examples decreases after applying the network constraints as shown in Figure 4.5 and Figure 4.6. This means that several of the positive examples given as answers by the methods are conflicting with false positive examples, hence both candidates are removed from the network of answers. We notice in Figure 4.6, the Business and Book datasets had an increase in the training size due to network constraints *deriving* a new matching using the cycle constraint (if A is matched to B and B is matched to C, we can derive that A is matched to C).

In contrast to the number of positive examples, the quality (measured by precision) of the set of *filtered* answers is better than the *unfiltered* answers. We measured the precision in the sets of answers and the results are shown in Figure 4.7 and Figure 4.8. By removing the uncertain candidates, we increase the probability of good training examples which, we believe, lead to better models learned.

Recalling Table 4.2, we notice that the `SF` answers break less constraints than `COMA`'s. This fact has a direct implication here, as the lower the number of constraints violated the smaller is the difference between *filtered* and *unfiltered* sets

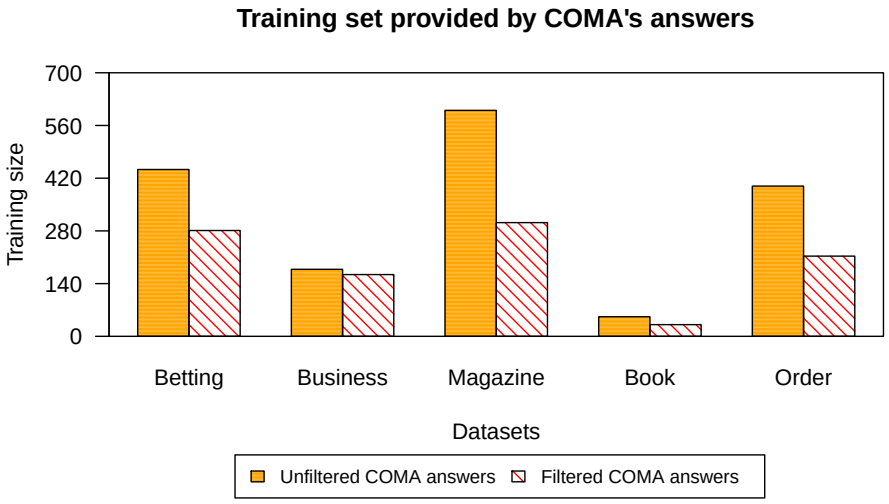

Figure 4.5: The amount of positive examples automatically acquired by taking COMA's answers.

of matches.

The network constraints allows us to automatically get negative examples as well. As the negative examples are the great majority of candidates, there are only few mislabels in that set and they do not impact in the learning process.

4.3.4 Training with Filtered Instances

The next experiment was designed to verify if filtered training sets in fact lead to better models learned. As performed in Section 4.3.1, `RF4SM-B` receives as training the matching network answers from a base system. This time, however, the training set provided is filtered by the network constraints presented in Section 2.2. Any pair of inconsistent matches are discarded from the set of examples. The base systems and `RF4SM` were used with the same settings as presented in previous experiments.

We present the average precision, recall and F1-scores of `RF4SM-B` when using

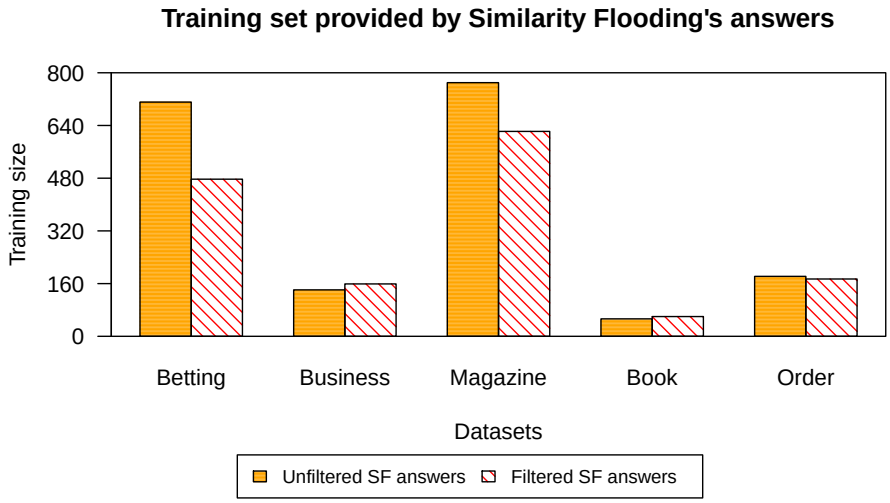

Figure 4.6: The amount of positive examples automatically acquired by taking Similarity Flooding's answers.

filtered answers from `COMA` in Figure 4.9 and `SF` in Figure 4.10. We notice that, when trained with filtered datasets, `RF4SM-B` achieved better results of precision and F1-score. This corroborates with the hypothesis that better learning sets generate better models. Also, we notice that not only the precision was improved, but the improvement was not in detriment of recall, as *filtered*-`RF4SM-B` achieved similar results of recall compared to the *unfiltered* version. This proves that the filters help to generate accurate models and they can generalize true matchings.

We can now establish that the *filtered* version of `RF4SM-B` is the more suitable to the schema matching networks problem. In the next section, we compare `RF4SM-B` with previous baselines to see if it is worth to automatically gather examples and filter them to use in a supervised method.

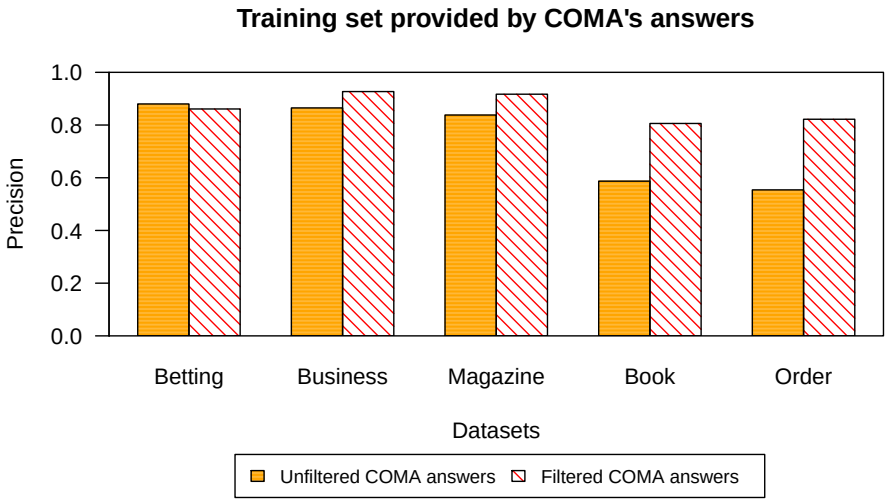

Figure 4.7: Precision of the positive examples given by COMA before and after appying the constraints filter.

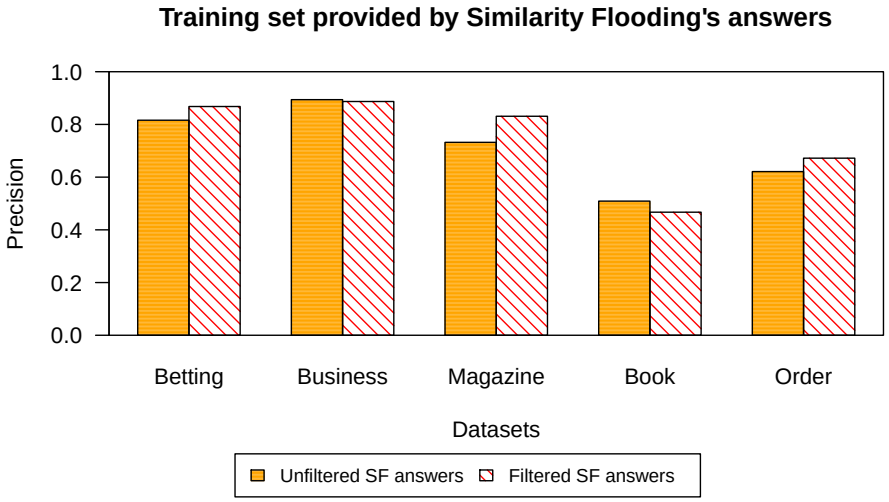

Figure 4.8: Precision of the positive examples given by Similarity Flooding before and after appying the constraints filter.

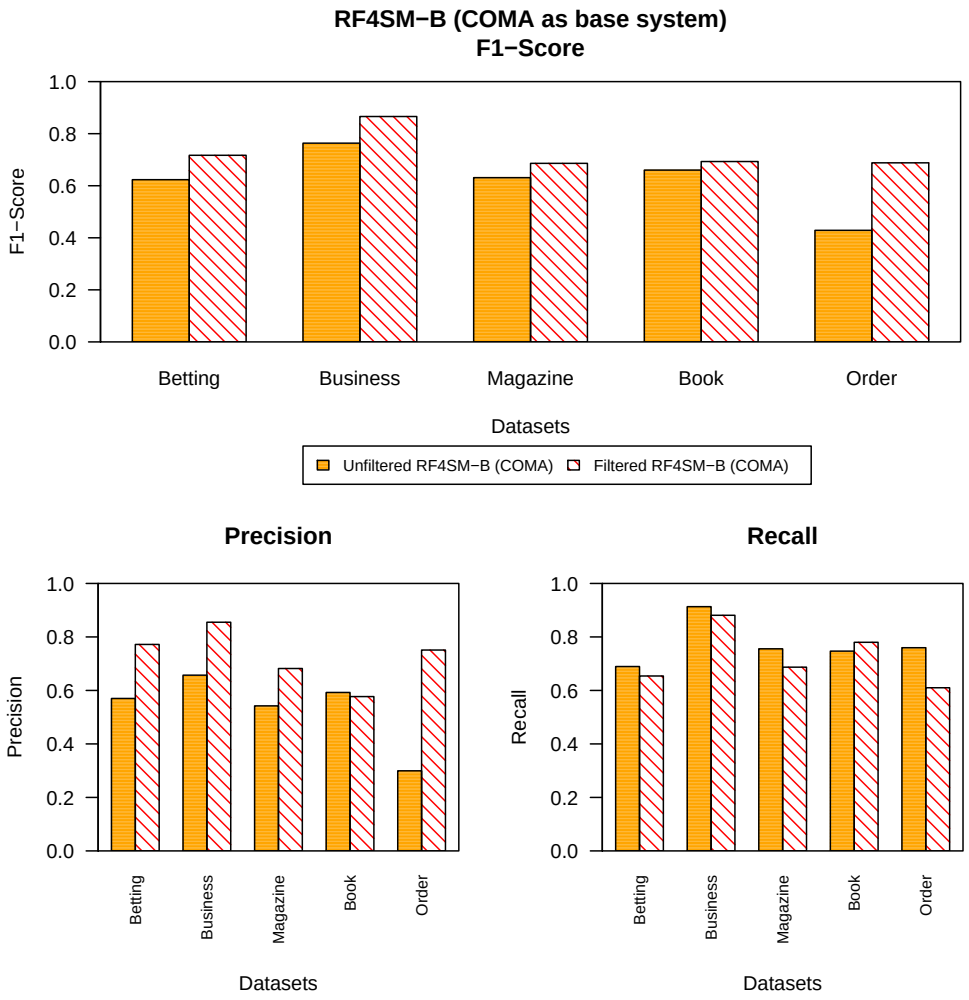

Figure 4.9: Average results obtained by `RF4SM-B` using the different training sets taken from `COMA`'s answers.

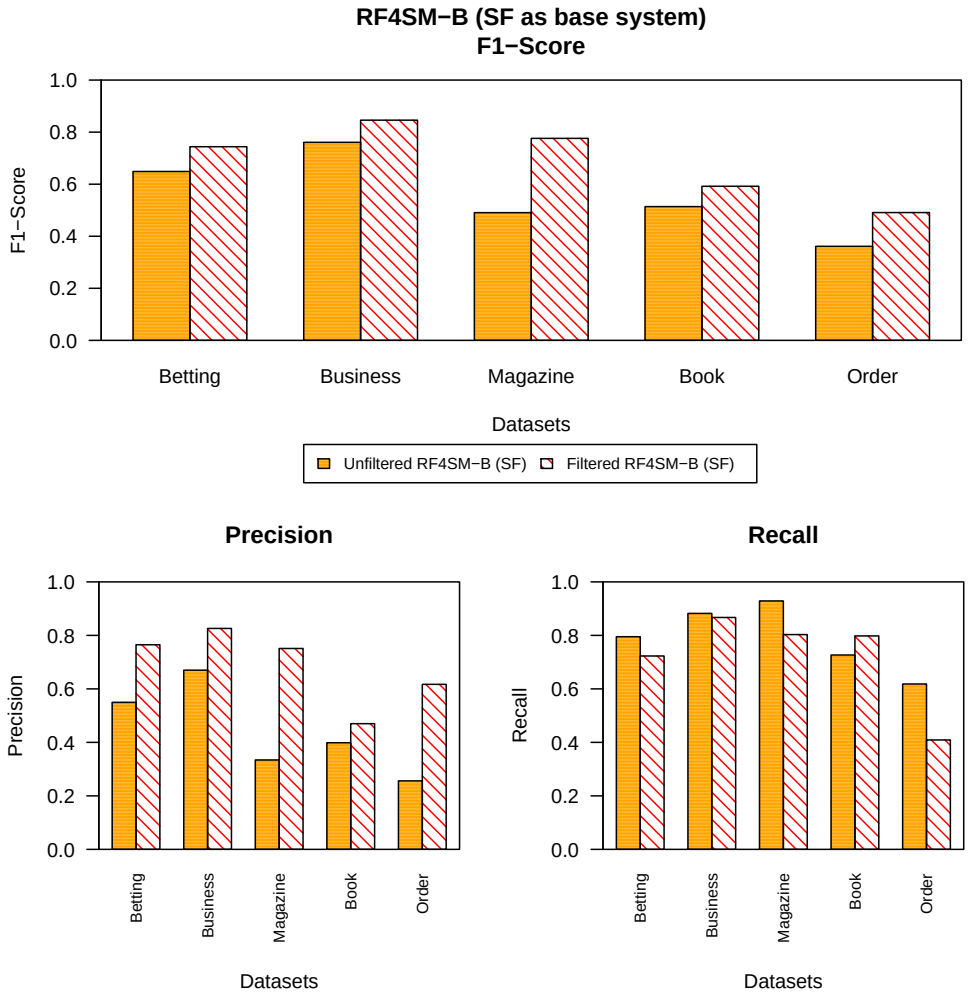

Figure 4.10: Average results obtained by `RF4SM-B` using the different training sets taken from `Similarity Flooding`'s answers.

4.3.5 RF4SM-B x Baselines

Once established the filtered set as the most suitable to our method, we compare RF4SM-B (both *Filtered* and *Unfiltered*) to the baselines. All of the strategies do not require any effort from a user, although RF4SM-B is a semi-supervised method. 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 base system and the F1-scores achieved are depicted in Figure 4.11. The second run uses Similarity Flooding as base system, the results are reported in Figure 4.12.

RF4SM-B can achive better results of F1-score in the majority of the datasets. On average, it outperforms both of the base systems, COMA and SF, and RF4SM whilst making use of 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 nor guaranteed to be 100% correctly labeled.

Results of precision and recall achieved by RF4SM-B in each dataset are shown in Figure 4.13 and Figure 4.14. We notice that precision when taking different base systems was not much different except in Book and Order datasets. Recall Figure 4.7 and Figure 4.8, these two datasets were the ones with big differences in training set precisions (the set given by COMA with higher precision than SF set).

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

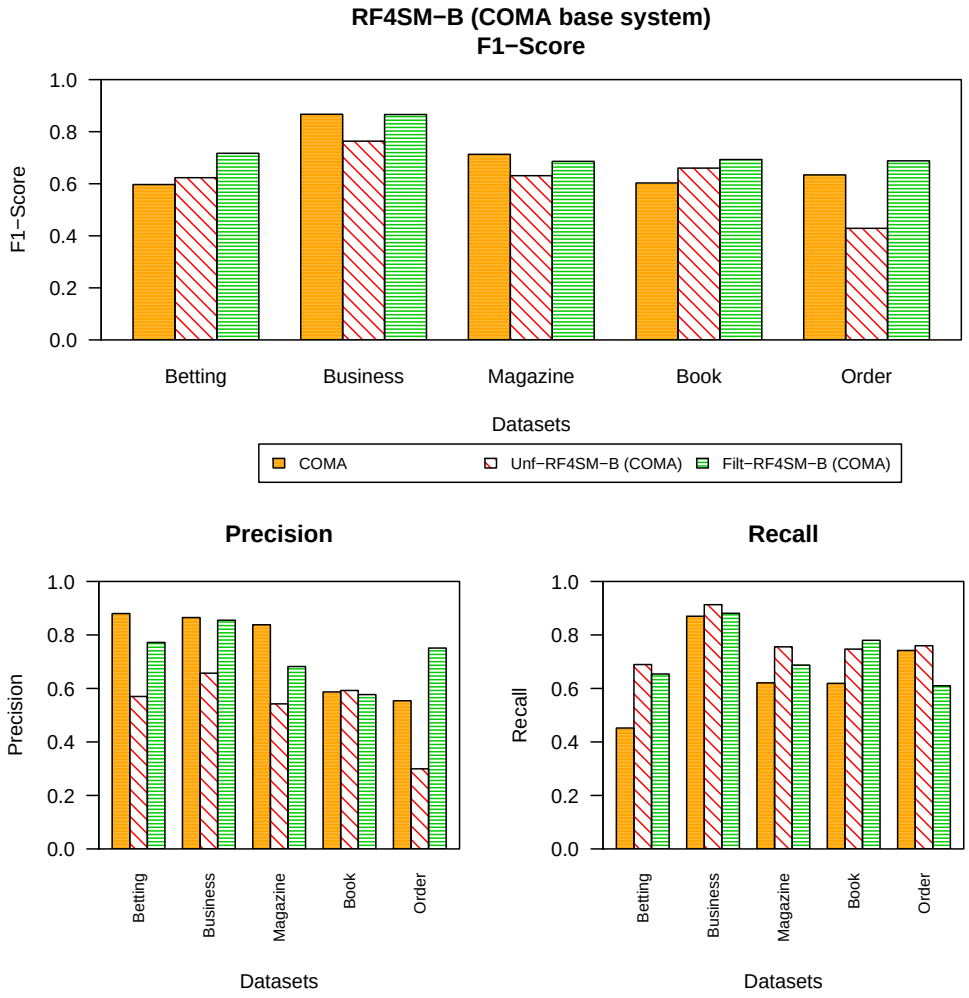

Figure 4.11: `RF4SM-B` (*Unfiltered* and *Filtered*) compared to `RF4SM` (when using `COMA` as the base system).

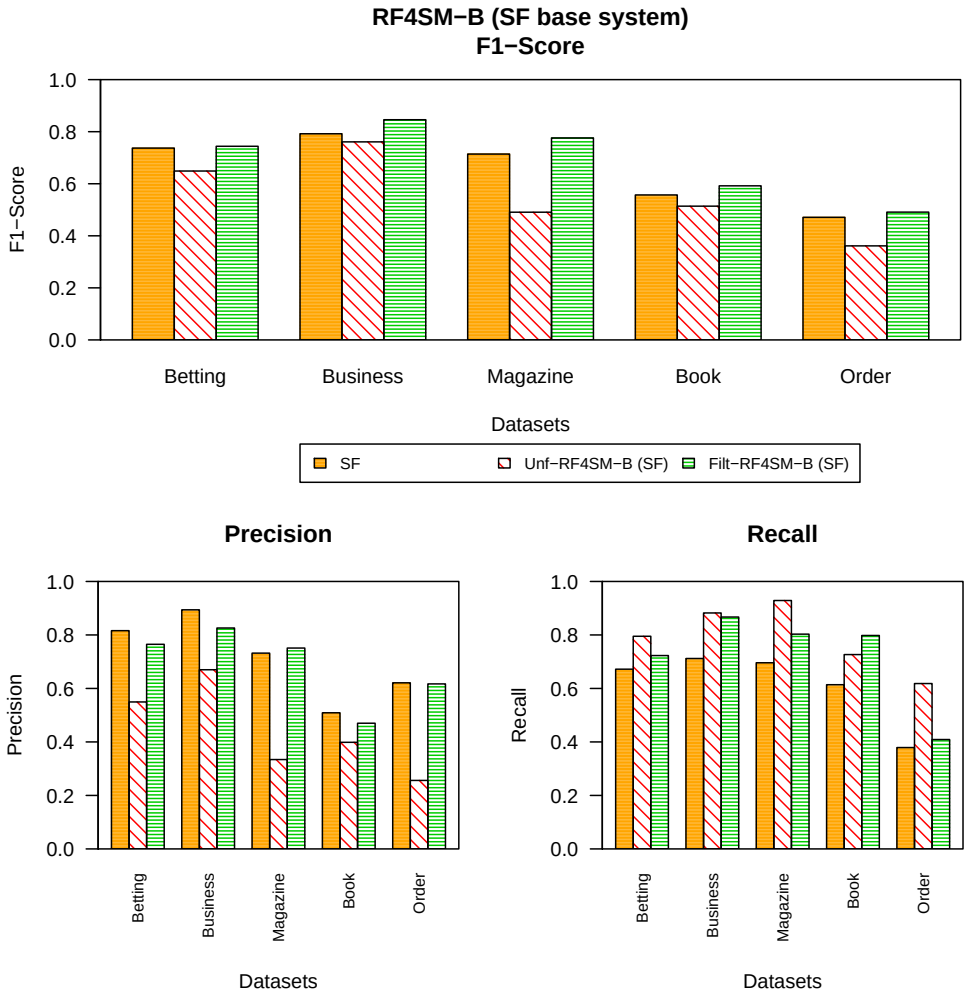

Figure 4.12: `RF4SM-B` (*Unfiltered* and *Filtered*) compared to `RF4SM` (when using `SF` as the base system).

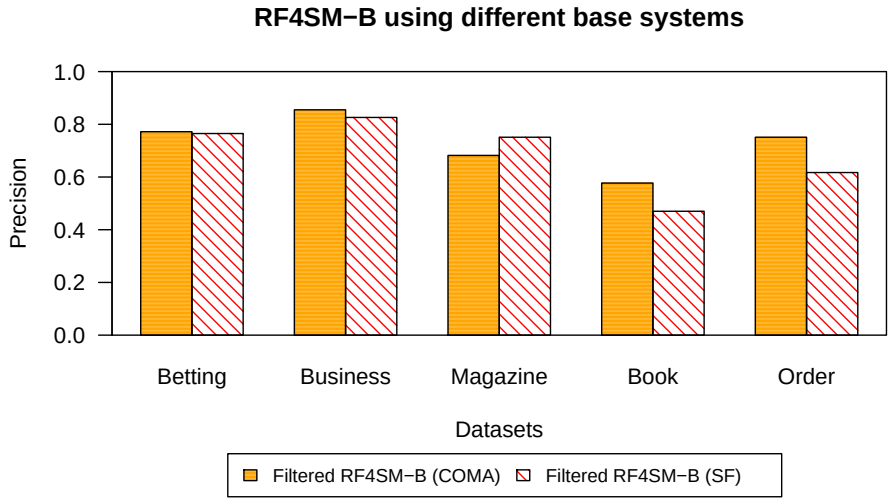

Figure 4.13: Precision achieved by `RF4SM-B` when running different base systems.

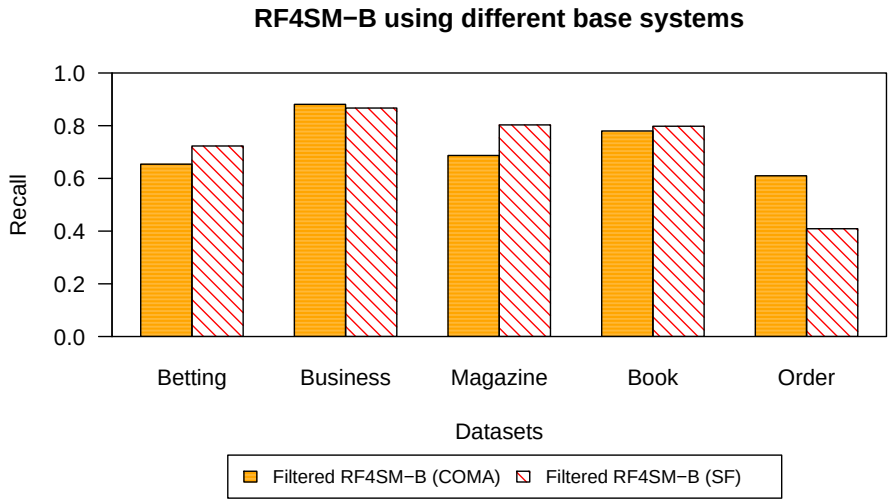

Figure 4.14: Recall achieved by `RF4SM-B` when running different base systems.

4.4 RF4SM-B-Rec: Experimental Evaluation

As observed in Section 4.3, `RF4SM-B` managed to train an ensemble of random trees by gathering a training set without asking the user for labels. However, the user can still be involved in the matching process by reviewing answers from a method, this process is called *Reconciliation*. The next set of experiments were designed to verify if the reconciliation brings improvements to the results and how great is the difference to the original matching.

Setup. In this set of experiments the methods were evaluated in all five datasets previous presented in Table 4.1. 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 remain the same as previous experiments. In this study, the user is considered to give only reliable assertions, i.e., if they confirm a candidate as a `TRUE` matching, then we can assume the matching is `TRUE`. The same can be considered to `FALSE` matchings. As there is no optimization aiming the minimum user-effort made, the order in which matching candidates are preseted for reviewing do not affect the outcome.

4.4.1 Reconciliation Benefits

As we consider the user will always give the correct feedback about a candidate matching, it is clear that they will always improve the results of a method (by removing false positives from the answers generated by a method). Thus, we decided to measure the improvement granted by the user when evaluating the list of straight answers of a method and when they evaluate the network of those answers, keeping in mind the amount of effort made in each case. The difference

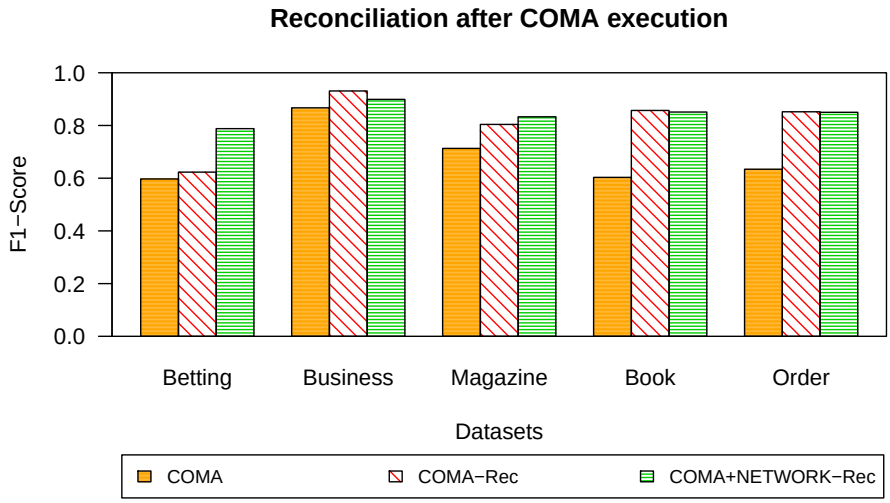

Figure 4.15: F1 scores reached after user feedback in two different scenarios based on `COMA`'s answers.

between the two situations is that, when considering the network of answers, the user approval or rejection of a matching candidate can affect the network and validate/invalidate other candidates leading to even better results, whereas in the other scenario the acceptance of an answer results in keeping the candidate in the set of answers whilst the rejection of an answer leads to its removal.

For each of the base methods (COMA and SF), we performed the reconciliation phase after the method execution and measured the scores of precision, recall and F1-score (reported as `BASE-METHOD-Rec`). Also, after the methods were ran, we built the network of answers of the methods and applied the network constraints to them, filtering the inconsistent answers. After that, we submitted the inconsistent answers to the user for approval/rejection and then measured the results (reported as `BASE-METHOD+NETWORK-Rec`).

We show results of F1-scores reached in this set of experiments. We used `COMA` and `Similarity Flooding` as base methods. Both results are shown in Figure 4.15

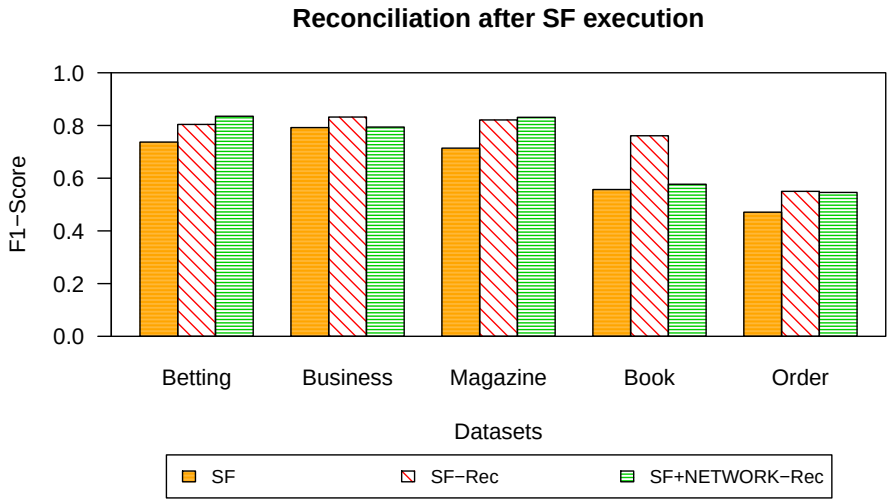

Figure 4.16: F1 scores reached after user feedback in two different scenarios based on `Similarity Flooding`'s answers.

(`COMA`) and Figure 4.16 (`Similarity Flooding`). As expected, all of the scenarios with the user input reached high levels of F1. When analizing precision and recall, the reconciliation helps to improve precision in all cases (as the user will reject false candidate matchings) and the recall can be higher when the network is used (as new undiscovered matchings can be found by applying the *cycle constraint*).

In many of the cases, one of two (or more) conflicting answers is accepted by the user, automatically rejecting the other. This elevates the importance of the user input as they virtually label two instances for the price of one. This effect is highlighted in Figure 4.17 and Figure 4.18, in the scenario `BASE METHOD-Rec`, the user has to look all of the answers given by a method in order to find false-positives. We stress that the number of answers of a method tends to grow as the number of schemas being matched get higher making it a more difficult task. When solving inconsistencies, the network constraints cut the user's job at least by half while keeping comparable levels of F1-Score, hence establishing the value

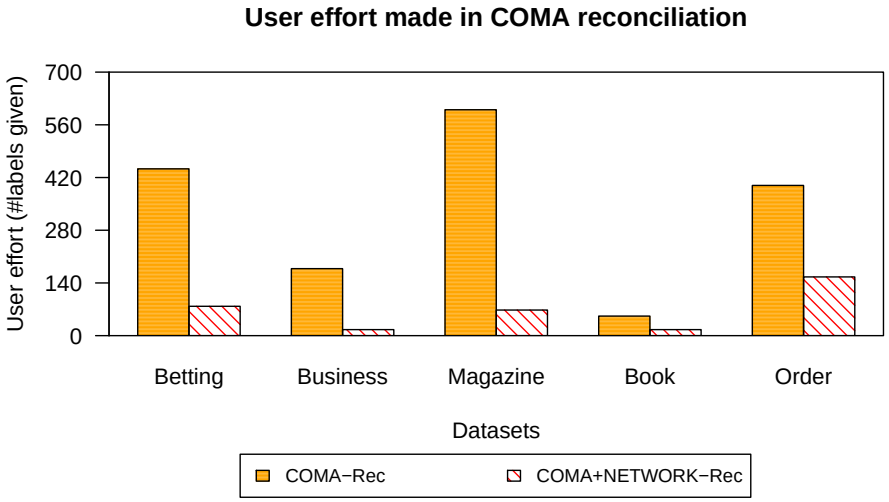

Figure 4.17: The effort made by the user in the reconciliation phase when labeling `COMA`'s candidate matchings.

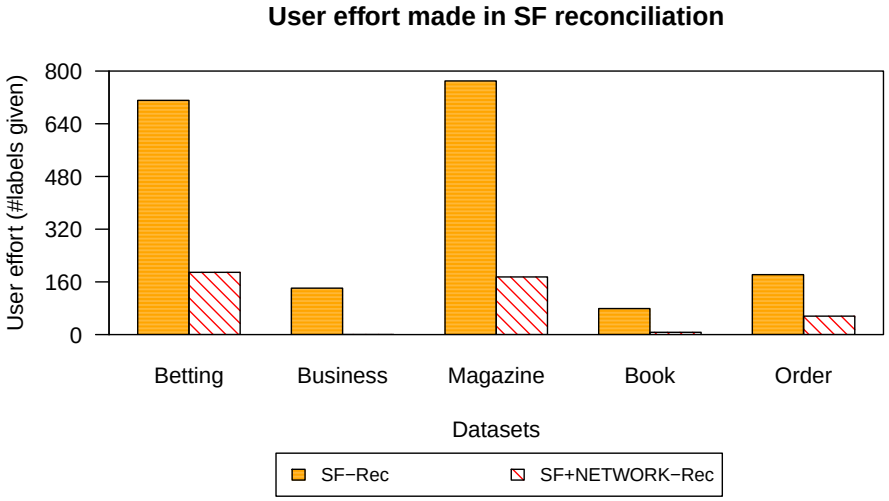

Figure 4.18: The effort made by the user in the reconciliation phase when labeling `Similarity Flooding`'s candidate matchings.

of these rules when considering the task of matching schemas.

4.4.2 RF4SM-B-Rec Results

After successfully acquiring training instances sparing the user of the job of labelling 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 of 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 removing it from the pool of answers (similar to the experiment in Section 4.4.1). After the reconciliation, we measured precision, recall and F1-scores of the methods.

We show the F1-scores reached in Figure 4.19 when we reconciliate the network of answers of `RF4SM-B` when it acquires automatic answers from `COMA`. On average, the reconciliation from `RF4SM-B` reaches F1-score of 0.83 against score of 0.81 from `COMA`'s reconciliation. However, `COMA`'s reconciliation requires way more user intervention as shown in Figure 4.21. Besides similar performance considering the F1-scores, from the point of view of a expert that will perform manually the reconciliation, it is most advantageous to go with `RF4SM-B` as its task is reduced by 5 times.

We also performed a similar experiment considering the automatic learning set will be gathered taking Similarity Flooding's answers. The F1-scores reached

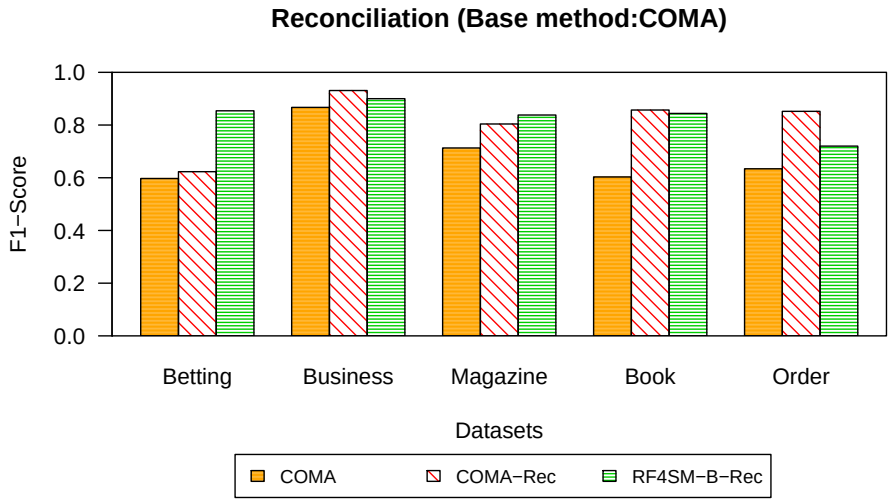

Figure 4.19: F1 scores achieved when performing reconciliation at the end of the process.

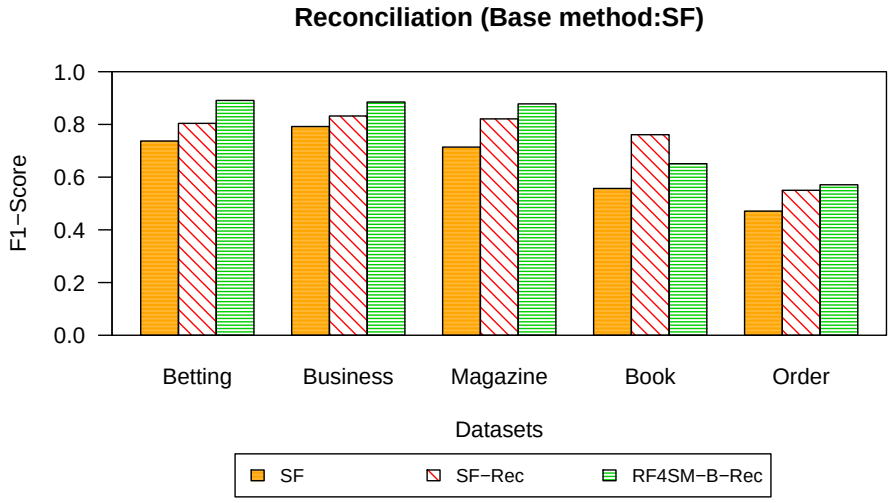

Figure 4.20: F1 scores achieved when performing reconciliation at the end of the process.

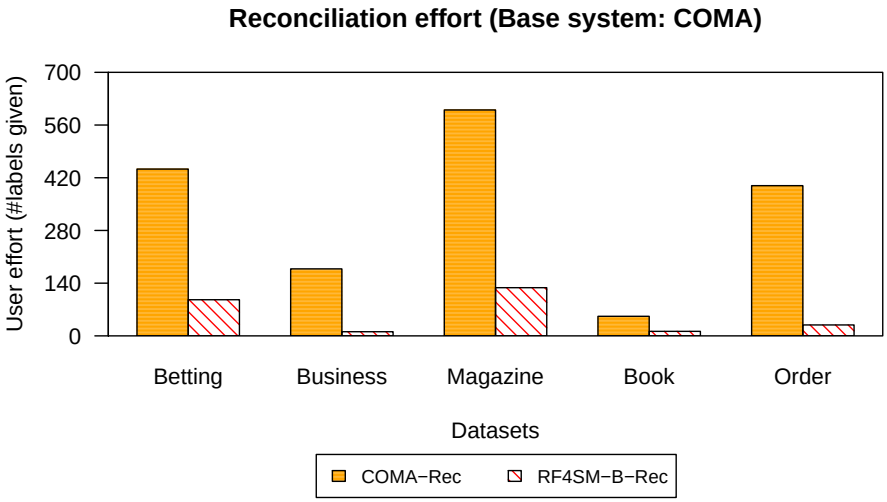

Figure 4.21: The user effort required in the reconciliation of the network of answers generated by COMA and RF4SM-B.

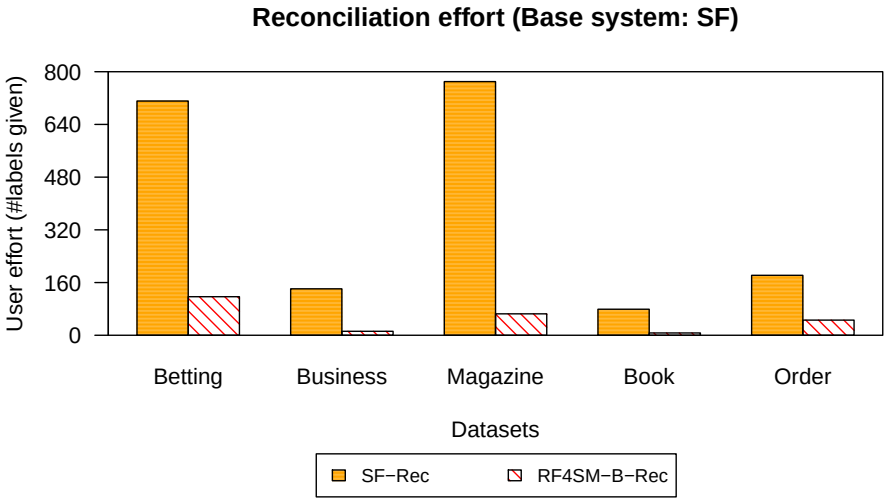

Figure 4.22: The user effort required in the reconciliation of the network of answers generated by SF and RF4SM-B.

are show in Figure 4.20. On average, `RF4SM-B` achieved F1-score of 0.77 against 0.75 from Similarity Flooding. The behavior was similar to the observed in the COMA's experiment, the user had to label a bigger amount of Similarity Flooding's answers to reach comparable F1-score. The amount of labels given by the user in the reconciliation task is given in Figure 4.22. In this experiment, the user effort also was reduced by approximately 6 times.

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

Finally, we compare results of `RF4SM-B` and `RF4SM-B-Rec` to see how much the reconciliation can improve the result of our method. We show the results of precision, recall and F1-score for the methods using `COMA` as the base system in Figure 4.23. As expected, the reconciliation process lead to better results achieving an average F1-score of 0.83 while asking for a low number of labels to the user (55 per dataset on average).

The same behavior can be observed when analizing results of the method when it uses `SF` as the base system (Figure 4.23). On average, `RF4SM-B-Rec` achieved F1-score of 0.77 and the user labeled 49 candidates per dataset leading us to believe that our method can be applied to any automatic matching system as a black box. Also by comparing the results of both base systems, the better the base system, the better will be the boosting provided by `RF4SM-B-Rec`.

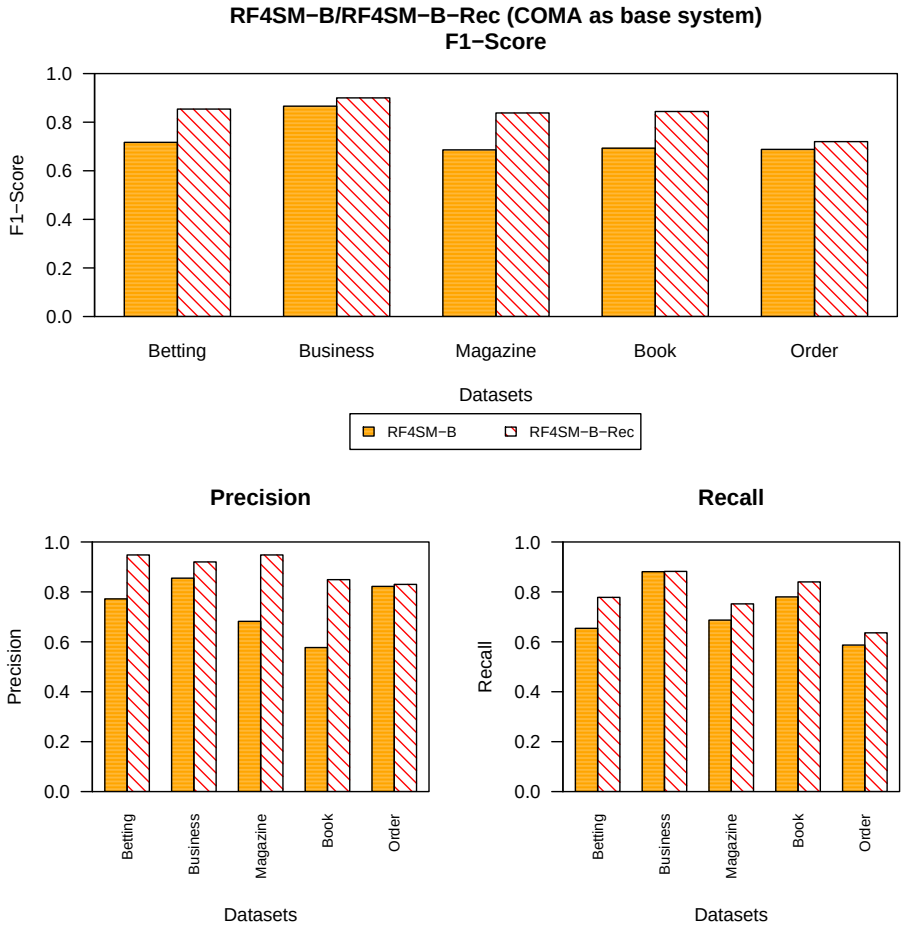

Figure 4.23: Comparative between `RF4SM-B` and `RF4SM-B-Rec` in all datasets using `COMA` as the base system.

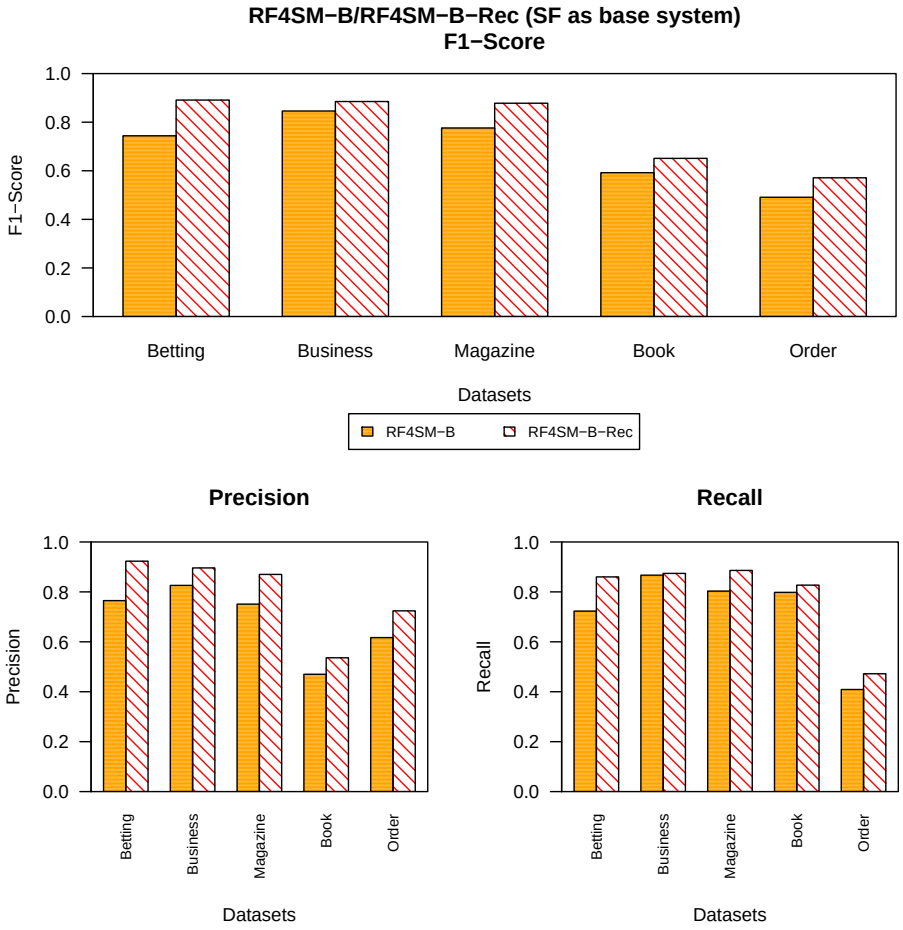

Figure 4.24: Comparative between `RF4SM-B` and `RF4SM-B-Rec` in all datasets using `SF` as the base system.

4.5 Summing up

We gathered the average F1-score of methods discussed in past sections and divided them in groups according to the type of user interaction in the matching task, they are all shown in Figure 4.25.

The first group "Exhaustive User Effort" contains the supervised method `RF4SM` that was run with different sizes of training. This group runs after a training set is provided. The training set used is labeled by an expert and contains a massive amount of examples. Although `RF4SM` can achieve higher results of precision than other methods, it required over 3000 training examples (in the smallest dataset) which is not suitable to our requirements in a real application.

The group "No User Effort" contains the heuristic methods (`COMA` and `SF`) and the semi-supervised boosting technique (`RF4SM-B`). None of the methods use any input from an expert (despite they can have tuned parameters) during the execution. Tipically these methods matching answers are reviewed by the expert after the execution. Besides taking into consideration the hierarchical aspect of the schemas, `COMA` and `SF` do not consider the network constraints, hence they rely on their heuristics that prioritize precision while `RF4SM-B` can create more generalized models that lack on precision in favor of recall, but they can be validated later.

The group "Some User Effort" has the reconciliation stage of all methods aforementioned. 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 great number of answers can be automatically invalidated. In addition, the network constraints help to gather a most reliable automatic training set of examples thus generating

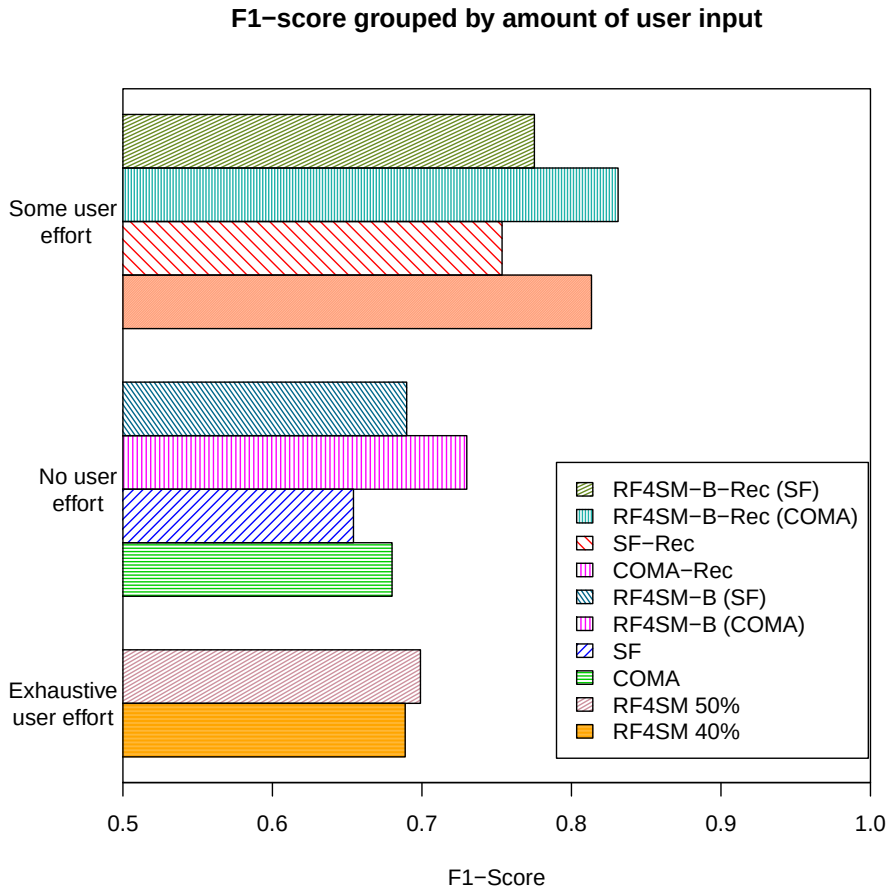

Figure 4.25: Average F1-score achieved by methods grouped by the amount of user participation in the matching task.

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 4 times in each dataset and using each of the base systems. `RF4SM-B-Rec` was able to achieve slightly higher values of F1-score while reducing the effort needed to validate the final matching answers and potentially can be used with any base system as a black box.

Chapter 5

Conclusions and Future Work

The schema matching problem has been studied for many years, in result of that, many research and commercial initiaves have been proposed. However, none of them fully solves the problem. In addition, the most effective approaches rely on user tunning or user assertion of generated matches.

With the integration of data being more present and being required by more applications, the schema matching networks problem has surfaced. In this setting, instead of matching a pair of schemas (the *classic schema matching*), it matches several schemas together. The schemas should have connections with all of the other schemas and must respect network integrity constraints.

In this work, we presented a study to apply machine learning techniques to the schema matching networks problem. To the best of our knowledge, this is the first piece of work that considers matching a network of schemas while respecting network integrity constraints. These constraints define rules to guarantee that the matching network remains consistent. Also, they might be used to help prune erroneous matches generated by our models.

First, we presented `RF4SM`, which is the method that uses machine learning algorithms to find correspondences between several schemas. In our experiments, we showed that the method is suitable for the problem of matching networks. However, it relies on a large number of labeled training examples which may not be available for every task. The method still reaches the highest average precision value: 0.70.

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 to generate synthetic training examples. Even with mislabeled examples, the method could train models and generate a large number of correct matches. We showed how the network constraints can be used to clean incorrect training examples and the impact they have on the final results. On average, *Filtered*-`RF4SM-B` achieved F1-score around 0.73.

Finally, we present `RF4SM-B-Rec`, which addresses the Schema Matching Reconciliation problem. In this task, the user is invoked to assert generated matchings by methods. They can accept or reject them. As previous methods do not consider integrity constraints, they miss out in a valueable source of information to prune bad correspondences. In our experiments, we show that `RF4SM-B` results can be even higher with the reconciliation process. We also show that F1-scores achieved are higher than other methods whilst asking for a lower number of labels. The experiments show that the user effort in the task can be reduced 6 times.

As future directions of our work, we recognize that there are still room for improvement: as evidenced by Hung et. al [Hung et al., 2014], the order the user assert 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 advances in the machine learning aspect can be made. In this work 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 yield large amount of data which is one of the challenges of the schema matching network problem.

Bibliography

[Aberer et al., 2003] Aberer, K., Cudre-Mauroux, P., and Hauswirth, M. (2003). Start making sense: The chatty web approach for global semantic agreements. *Journal of Web Semantics*, 1(1):89–114.

[Alani and Saad, 2017] Alani, H. and Saad, S. (2017). Schema matching for large-scale data based on ontology clustering method. *International Journal on Advanced Science, Engineering and Information Technology*, 7(5):1790–1797.

[Aumueller et al., 2005] Aumueller, D., Do, H.-H., Massmann, S., and Rahm, E. (2005). Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 906–908. ACM.

[Bellahsene et al., 2011] Bellahsene, Z., Bonifati, A., and Rahm, E., editors (2011). *Schema Matching and Mapping*. Springer.

[Bernstein et al., 2011] Bernstein, P. A., Madhavan, J., and Rahm, E. (2011). Generic schema matching, ten years later. *PVLDB*, 4(11):695–701.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Bonifati and Velegrakis, 2011] Bonifati, A. and Velegrakis, Y. (2011). Schema matching and mapping: from usage to evaluation. In *Proc. of the 14th Intl. Conf. on Extending Database Technology*, pages 527–529.

[Cruz et al., 2009] Cruz, I. F., Antonelli, F. P., and Stroe, C. (2009). Agreementmaker: efficient matching for large real-world schemas and ontologies. *Proc. VLDB Endowment*, 2(2):1586–1589.

[Cudré-Mauroux et al., 2006] Cudré-Mauroux, P., Aberer, K., and Feher, A. (2006). Probabilistic message passing in peer data management systems. In *ICDE*, page 41. IEEE Computer Society.

[de Carvalho et al., 2013] de Carvalho, M. G., Laender, A. H. F., Gonçalves, M. A., and da Silva, A. S. (2013). An evolutionary approach to complex schema matching. *Information System*, 38(3):302–316.

[Do and Rahm, 2002] Do, H.-H. and 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*, VLDB '02, pages 610–621. VLDB Endowment.

[Doan et al., 2001] Doan, A., Domingos, P., and Halevy, A. Y. (2001). Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record*, volume 30, pages 509–520. ACM.

[Doan et al., 2000] Doan, A., Domingos, P., and Levy, A. (2000). Learning source descriptions for data integration. In *Proceedings of the Third International Workshop on the Web and Databases*, WebDB'2000, pages 81–86.

[Doan et al., 2012] Doan, A., Halevy, A. Y., and Ives, Z. G. (2012). *Principles of Data Integration*. Morgan Kaufmann.

[Drumm et al., 2007] Drumm, C., Schmitt, M., Do, H.-H., and Rahm, E. (2007). Quickmig: Automatic schema matching for data migration projects. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 107–116, New York, NY, USA. ACM.

[Duchateau and Bellahsene, 2010] Duchateau, F. and Bellahsene, Z. (2010). Measuring the quality of an integrated schema. In *Proceedings of the 29th International Conference on Conceptual Modeling*, ER'10, pages 261–273, Berlin, Heidelberg. Springer-Verlag.

[Duchateau et al., 2008] Duchateau, F., Bellahsene, Z., and Coletta, R. (2008). A flexible approach for planning schema matching algorithms. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems: OTM 2008*, pages 249–264, Berlin, Heidelberg. "Springer Berlin Heidelberg.

[Duchateau et al., 2009] Duchateau, F., Coletta, R., Bellahsene, Z., and Miller, R. J. (2009). (not) yet another matcher. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1537–1540, New York, NY, USA. ACM.

[Freund and Schapire, 1997] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139.

[Gal, 2006] Gal, A. (2006). Why is schema matching tough and what can we do about it? *SIGMOD Rec.*, 35(4):2–5.

[Gal and Sagi, 2010] Gal, A. and Sagi, T. (2010). Tuning the ensemble selection process of schema matchers. *Inf. Syst.*, 35(8):845–859.

[Hall et al., 2009] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

[He and Chang, 2003] He, B. and Chang, K. C.-C. (2003). Statistical schema matching across web query interfaces. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 217–228, New York, NY, USA. ACM.

[He et al., 2004] He, B., Chang, K. C.-C., and Han, J. (2004). Discovering complex matchings across web query interfaces: A correlation mining approach. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 148–157, New York, NY, USA. ACM.

[Ho, 1995] Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1.

[Hung et al., 2015] Hung, N. Q. V., Tam, N. T., Chau, V. T., Wijaya, T. K., Mikl., Aberer, K., Gal, A., and Weidlich, M. (2015). Smart: A tool for analyzing and reconciling schema matching networks. In *ICDE*, pages 1488–1491. IEEE.

[Hung et al., 2013] Hung, N. Q. V., Tam, N. T., Miklós, Z., and Aberer, K. (2013). *On Leveraging Crowdsourcing Techniques for Schema Matching Networks*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Hung et al., 2014] Hung, N. Q. V., Tam, N. T., Miklós, Z., Aberer, K., Gal, A., and Weidlich, M. (2014). Pay-as-you-go reconciliation in schema matching networks. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 220–231.

[Lee et al., 2007] Lee, Y., Sayyadian, M., Doan, A., and Rosenthal, A. S. (2007). etuner: Tuning schema matching software using synthetic scenarios. *The VLDB Journal*, 16(1):97–122.

[Li et al., 2005] Li, Y., Liu, D.-B., and Zhang, W.-M. (2005). Schema matching using neural network. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 743–746.

[Madhavan et al., 2005] Madhavan, J., Bernstein, P. A., Doan, A., and Halevy, A. (2005). Corpus-based schema matching. In *21st International Conference on Data Engineering (ICDE'05)*, pages 57–68.

[Madhavan et al., 2001] Madhavan, J., Bernstein, P. A., and Rahm, E. (2001). Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 49–58, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Melnik et al., 2002] Melnik, S., Garcia-Molina, H., and Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm. In *Proceedings of Eighteenth International Conference on Data Engineering*, San Jose, California.

[Miller et al., 2009] Miller, F., Vandome, A., and McBrewster, J. (2009). *Levenshtein Distance*. VDM Publishing.

[Ngo and Bellahsene, 2012] Ngo, D. and 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*, EKAW'12, pages 421–425, Berlin, Heidelberg. Springer-Verlag.

[Nguyen et al., 2011] Nguyen, H., Fuxman, A., Paparizos, S., Freire, J., and Agrawal, R. (2011). Synthesizing products for online catalogs. *Proc. VLDB Endow.*, 4(7):409–418.

[Nguyen et al., 2013] Nguyen, H. Q. V., Luong, X. H., Miklós, Z., Quan, T. T., and Aberer, K. (2013). *Collaborative Schema Matching Reconciliation*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Nguyen, 2014] Nguyen, Q. V. H. (2014). Reconciling schema matching networks.

[Peukert et al., 2011] Peukert, E., Eberius, J., and Rahm, E. (2011). Amc - a framework for modelling and comparing matching systems as matching processes. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 1304–1307. IEEE Computer Society.

[Popa et al., 2002] Popa, L., Hernadez, M. A., Velegrakis, Y., Miller, R., Naumann, F., and Ho, H. (2002). Mapping xml and relational schemas with CLIO. In *Proc. of the 18th Intl. Conf. on Data Engineering*, pages 0–498.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Rahm and Bernstein, 2001] Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350.

[Reis et al., 2017] Reis, D. G., Carvalho, R. N., Carvalho, R. S., and Ladeira, M. (2017). Two-phase parallel learning to identify similar structures among relational databases. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1020–1023.

[Rodrigues et al., 2015] Rodrigues, D., da Silva, A. S., Rodrigues, R., and dos Santos, E. (2015). Using active learning techniques for improving database schema matching methods. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, pages 1–8.

[Rong et al., 2012] Rong, S., Niu, X., Xiang, E. W., Wang, H., Yang, Q., and Yu, Y. (2012). A machine learning approach for instance matching based on similarity metrics. In Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J. X., Hendler, J., Schreiber, G., Bernstein, A., and Blomqvist, E., editors, *The Semantic Web – ISWC 2012*, pages 460–475, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Shiang et al., 2008] Shiang, W.-J., Chen, H.-C., and Rau, H. (2008). An intelligent matcher for schema mapping problem. In *2008 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3172–3177.

[Su et al., 2006] Su, W., Wang, J., and Lochovsky, F. (2006). Holistic schema matching for web query interfaces. In *Proceedings of the 10th International Conference on Advances in Database Technology*, EDBT'06, pages 77–94, Berlin, Heidelberg. Springer-Verlag.

[Toan et al., 2018] Toan, N. T., Cong, P. T., Thang, D. C., Hung, N. Q. V., and Stantic, B. (2018). Bootstrapping uncertainty in schema covering. In Wang, J., Cong, G., Chen, J., and Qi, J., editors, *Databases Theory and Applications*, pages 336–342, Cham. Springer International Publishing.

[Walker and Duncan, 1967] Walker, S. H. and Duncan, D. B. (1967). Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 54(1-2):167–179.

[Williams, 2010] Williams, P. K. (2010). *A Clustering Rule Based Approach for Classification Problems*. PhD thesis, Auburn, AL, USA. AAI3430661.