The VLDB Journal (2023) 32:305–324
https://doi.org/10.1007/s00778-022-00743-3

REGULAR PAPER

Fine-grained semantic type discovery for heterogeneous sources using clustering

Federico Piai[1] · **Paolo Atzeni**[1] · **Paolo Merialdo**[1] · **Divesh Srivastava**[2]

Received: 4 May 2021 / Revised: 19 December 2021 / Accepted: 22 March 2022 / Published online: 17 May 2022

Abstract
We focus on the key task of semantic type discovery over a set of heterogeneous sources, an important data preparation task. We consider the challenging setting of multiple Web data sources in a vertical domain, which present sparsity of data and a high degree of heterogeneity, even internally within each individual source. We assume each source provides a collection of entity specifications, i.e. entity descriptions, each expressed as a set of attribute name-value pairs. Semantic type discovery aims at clustering individual attribute name-value pairs that represent the same semantic concept. We take advantage of the opportunities arising from the redundancy of information across such sources and propose the iterative RAF- STD solution, which consists of three key steps: (i) a Bayesian model analysis of overlapping information across sources to match the most locally homogeneous attributes; (ii) a tagging approach, inspired by NLP techniques, to create (virtual) homogeneous attributes from portions of heterogeneous attribute values; and (iii) a novel use of classical techniques based on matching of attribute names and domains. Empirical evaluation on the DI2KG and WDC benchmarks demonstrates the superiority of RAF- STD over alternative approaches adapted from the literature.

Keywords Data integration · Data preparation · Semantic typing · Semantic type Discovering · Schema matching · Big data · Web data

1 Introduction

Many applications need to exploit data from multiple Web sources, each providing detailed descriptions about entities of vertical domains (e.g. products, people, companies) [14,16–18,20,25,31,33]. Each source might be the result of an extraction process from a website, or the result of a query against an API. As Web sources are usually noisy and heterogeneous, several data preparation tasks [12], including data cleaning [1,13] and entity identifier extraction [4,35], must be performed prior to passing the data to downstream applications.

Given a collection of sources, an important data preparation problem that has been recently addressed in the literature is that of discovering the semantic types, i.e. clustering data items that represent instances of a semantic concept. So far, the problem has been tackled for strings [43], for tabular (relational) data [23,45], and for collections of structured datasets [34]. These proposals represent effective solutions to discover semantic types in several application scenarios. However, as we discuss in Sect. 2, their assumptions represent limitations when dealing with wild data from the Web because of the high degree of heterogeneity that arises not only across sources, but also within individual sources.

To illustrate the problem and its challenges, consider the example data shown in Fig. 1, which are derived from the DI2KG camera dataset, a recent data integration benchmark publicly available to the research community. [1]. The dataset provides about 30k camera specifications from 24 real Web sources, consisting of a total of 528k attribute name-value

✉ Paolo Merialdo
paolo.merialdo@uniroma3.it

Federico Piai
federico.piai@uniroma3.it

Paolo Atzeni
paolo.atzeni@uniroma3.it

Divesh Srivastava
divesh@research.att.com

[1] Department of Engineering, Roma Tre University, Rome, Italy

[2] AT & T Chief Data Office, Bedminster, NJ, USA

[1] http://di2kg.inf.uniroma3.it/datasets.html

S_1	S_2	S_3	S_4	S_5
s_{11} { Battery chemistry: Li-Ion, Battery model: NP-BN1, CPU: BionzX }	s_{21} { B type: Li-ion, IPU: BionzX }	s_{31} { Memory: 16, Battery: Li-Ion }	s_{41} { Memory: 16, Battery: NP-BN }	s_{51} { Megapixels: 16, Batt: included Li-Ion }
s_{12} { Battery chemistry: Ni-MH, Battery Model: FNB83, Processor: Xpeed3 }	s_{22} { B type: Ni-Mh, IPU: Xpeed3 }	s_{32} { Memory: 8, Battery: FNB83 }	s_{42} { Memory: 8, Battery: FNB83 }	s_{52} { Megapixels: 16, Batt: Ni-MH battery }
s_{13} { Battery chemistry: Li-Ion, Processor: BionzX, Video resolution: 1024x768 }	s_{23} { IPU: BionzX, Video format: 1024x768 }	s_{33} { Memory: 32, Battery: LP-E6N Li-Ion }	s_{43} { Memory: 16, Battery: LP-E6N }	s_{53} { Megapixels: 8, Batt: Li-Ion rechargeable }
s_{14} { CPU: Xpeed2, Video resolution: 1920x1024 }	s_{24} { IPU: Xpeed }	s_{34} { Memory: 16, Battery: Ni-MH, }	s_{44} { Battery Type: NI-MH }	s_{54} { Megapixels: 16, Batt: Ni-Cd rechargeable }
s_{15} { Processor: Xpeed, Video resolution: 800x600, Battery chemistry: Ni-Mh }	s_{25} { IPU: Xpeed2, Video format: 1920x1024, Megapixels: 8 MPX }	s_{35} { Battery: Li-Ion Pac LP-E6N }	s_{45} { Battery Type: Li-Ion }	s_{55} { Batt: Ni-MH available }
	s_{26} { Megapixels: 16 MPX, B type: LiPo, IPU: Digic8 }		s_{46} { Battery Type: Ni-Cd }	s_{56} { Batt: Ni-Cd rechargeable }

Linkage: $\{s_{11}, s_{21}, s_{31}, s_{41}\}$, $\{s_{12}, s_{22}, s_{32}, s_{42}, s_{52}\}$, $\{s_{13}, s_{23}, s_{33}, s_{43}, s_{53}\}$, $\{s_{14}, s_{25}\}$, $\{s_{35}, s_{45}\}$

Fig. 1 Running example: five sources with some entity specifications, and linkage information (each set represents a group of specifications that refer to the same entity)

pairs. Each source is a collection of entity specifications, where each entity specification consists of a set of attribute name and value pairs, represented as a JSON object.

Challenges Let us comment on the major forms of heterogeneity that we observed across sources as well as within individual sources in the DI2KG dataset. They represent challenging issues to the semantic type discovery problem.

- *Attribute-Name Heterogeneity* Attributes with the same semantic type can be represented by different names, even across specifications of the same source. For example, observe attribute names `CPU` and `Processor` in source S_1 in Fig. 1: they have the same meaning, yet some entity specifications use `CPU` and others `Processor`. Conversely, there are *overloaded* attribute names, i.e. there are attributes with the same name but different semantic types, even within the same source. Observe the `Battery` attribute in source S_3: in some entity specifications the value refers to the battery chemistry (s_{31}, in the example), in others to the battery model (s_{32}), or even to both (s_{33}).
- *Attribute-Value Heterogeneity* Even when a source does not have attribute-name heterogeneity, it may have values that use different representations across entities. Since Web data are typically created for human consumption, sources can introduce pieces of free text (usually to aid human users) within the values. In our example, consider the `Batt` attribute in source S_5 whose values are adorned with descriptive words, such as "battery" and "rechargeable" (which are indeed superfluous, as every Li-Ion or Ni-MH battery is rechargeable). Another form of attribute value heterogeneity is due to different granularities of representation: some values could be compositions of pieces of information that elsewhere—even in the same source—are represented by means of multiple attributes. In our example, consider the `Battery` attribute for the specifications s_{33} and s_{35}, whose values include both the battery type and the battery chemistry, without any underlying pattern. Finally, it is important to observe that Web data are noisy and that some sources can publish erroneous values; in our running example, s_{33} and s_{43} are linked, i.e. they represent the same entity, but they have different values for attribute `Memory` (32 and 16, respectively).
- *Attribute sparsity* The average number of distinct attribute names per source is 273. However, on average, each entity specification has 18 attributes: less than 7% of attribute names are used for each entity. If it were a relational database, we would say that 93% of attribute values are *null*. Concretely, each source presents a huge variety of attributes, many of which are provided only for few entity specifications, increasing the difficulty of discovering semantic types.

In this paper, we address the problem of discovering the semantic types of attributes in such a challenging scenario. Our goal is to create clusters of attributes of individual specifications with the same semantics: each cluster represents a semantic type. Figure 2 illustrates the output of our method for a subset of data in Fig. 1. Each cluster consists of a set of attributes of individual specifications, suitably represented as triples.

Our approach, RAF- STD (RAF—Semantic Type Discovery), has been developed in the context of RAF (Redundancy as Friend) [4,36], an ongoing research project that addresses the issue of end-to-end integration of entity specifications from multiple heterogeneous sources. The overall approach of the project leverages opportunities that arise from the redundancy of information among and across sources.

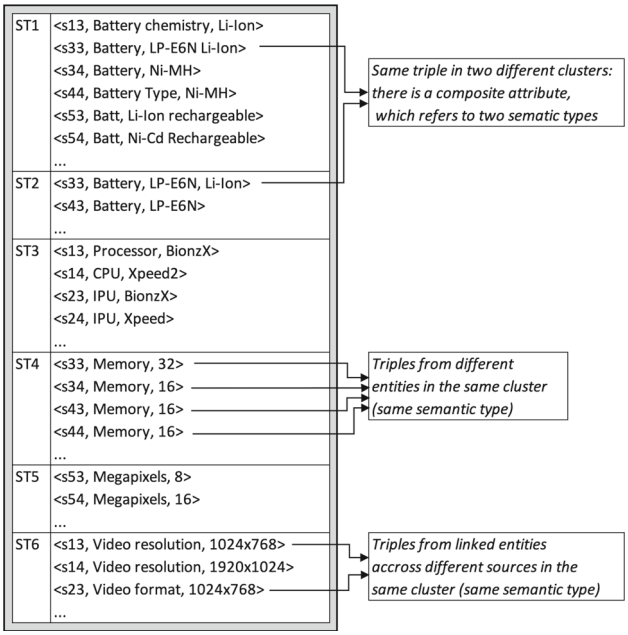

Fig. 2 Semantic type clusters for attributes of entity specifications s_{*3}, s_{*4} in Fig. 1. Notice that: (i) semantic type clusters may overlap: for example, $\langle s_{33}, \texttt{Battery}, \text{LP-E6N Li-Ion}\rangle$ appears in two clusters ($ST1$ and $ST2$), as its value is composite and refers to two semantic types (battery model and battery chemistry); (ii) a semantic type cluster contains triples from entity specifications referring to different entities: for example, the semantic type ST4 contains triples from s_{33}, s_{34}, s_{43}, s_{44}, but only s_{33} and s_{43} are linked (i.e. refer to the same entity); (iii) semantic type clusters may contain triples from linked entities across different sources: for example, in the semantic type cluster $ST6$, the triples $\langle s_{13}, \texttt{Video resolution}, 1024\text{x}768\rangle$ and $\langle s_{23}, \texttt{Video format}, 1024\text{x}768\rangle$ refer to the same entity (see linkage in Fig. 1) and belong to different sources (S_1 and S_3, respectively)

Opportunities In our setting, we can identify several of these opportunities:

- Many entities appear in different sources with shared values. This redundancy can be exploited for our purposes, based on the intuition that attributes from different sources with the same values for the same entity are likely to represent the same semantic type. We can, indeed, take advantage of the availability of entity matching solutions, such as those based on relational pre-trained transformers [10,28,42], or those based on unsupervised techniques [35,36,44] that effectively work even without needing to deal with attribute reconciliation.
- Although heterogeneity occurs even within sources, it is usually the case that at least a few attributes in a source tend to be homogeneous. A useful opportunity is to identify and first cluster such attributes into semantic types across sources.
- Some sources use different names for attributes that are overloaded in other sources. This represents an opportunity to resolve the semantic types of overloaded attribute names (e.g. S_1 uses different names for battery chemistry and battery model, and this allows the resolution of the overloading of `Battery` in S_3).
- Common meaningful values (e.g. "Li-Ion") appear in multiple specifications, for different entities, within and across sources, despite many kinds of heterogeneity. This redundancy can be exploited.[2]
- Given the number of sources, there is a lot of redundancy of information within and across them, despite the attribute sparsity. Redundancy across sources can be exploited as evidence of equivalence of semantic types.

We propose an iterative approach consisting of three key steps to solve the semantic typing problem by taking advantage of the above opportunities. The first step is based on a Bayesian model that analyses overlapping information across sources to match specification attributes that most likely have the same meaning, exploiting partial homogeneity that occurs inside sources. The second step is based on a tagging approach, inspired by NLP techniques [25], to create groups of *virtual* specification attributes from tagged portions of values not matching in the previous step. Matching and tagging are iterated, as they produce complementary evidence that reinforces each other. A final step processes the results of the iterations and aims at improving the created clusters by adding specification attributes that the previous steps could not merge (for example due to lack of linkage, or for differences in representation of values).

Paper Outline Section 2 discusses related work. Section 3 presents the problem setting and an overview of our solution. Section 4 describes the Bayesian approach to cluster the most similar entity specification attributes. Section 5 illustrates our tagging solution and the iterative algorithm to produce semantic typing. Section 6 reports on experiments with different datasets, under different conditions, and comparisons with other approaches. Finally, Sect. 7 concludes the paper with final remarks and future work.

2 Related work

Semantic type detection has been recently addressed in the literature. Related issues arise also in research on table to knowledge base matching and on schema matching. In the following, we discuss how our approach differs from these bodies of research. In our experimental evaluation (Sect. 6), we compare our solution to one representative for each of these.

[2] There are values that occur too often for multiple attributes (such as, "yes" and "no") and therefore cannot be considered to be meaningful of this exploitation, as we will see in Sect. 4.

Semantic Type Detection Sherlock [23] and SATO [45] adopt a supervised approach to associate each column of a relational table with a type from a fixed set of predefined semantic types (a restricted number of types derived from DBPedia properties [38]). Such a closed world approach represents a strong limitation as it prevents the identification of the diversity of types that are usually present in many vertical domains. Also, both these systems work at the column level, i.e. they associate a semantic type with a column as a whole, assuming that all the elements of the column refer to the same type. As we discussed in the Introduction, this assumption is not valid with Web data, which are characterized by sparsity of data and semantic heterogeneity of attribute names and values. Another drawback of Sherlock and SATO is the training effort, as they require a large set of tables with columns labeled with a semantic type.

Differently from Sherlock and SATO, D4 [34] adopts an unsupervised, data-driven approach: it does not rely on training data, and it does not have to refer to a predefined set of types. Given a collection of relational tables, D4 aims to identify sets of terms that represent values of the same semantic type. Also, D4 takes into account that columns might be heterogeneous and contain multiple types of values, but it assumes homogeneity of values.

In our experimental evaluation, we compare our solution to D4.

Table to Knowledge Base Matching Semantic type detection is also related to solutions for populating an existing knowledge base with facts extracted from tabular data, typically data extracted from HTML tables (see [46] for a survey).

A seminal work in this area is that developed by Limaye *et al.*, who proposed a supervised approach based on a probabilistic graphical model [29]. The Tabel system [7] builds on the Limaye *et al.*'s approach relying also on the co-occurrence of the elements in the table and in Wikipedia documents. The redundancy of information in tables and web documents has been exploited in other approaches, such as [11,41], that, differently from the previous ones, are unsupervised. However, these approaches assume a significant overlap between the table and the target knowledge base, and need a huge corpus of textual documents whose contents are related to the elements of the tables.

Among the proposals for table to knowledge base matching, T2K [39] represents an interesting solution: it assumes overlap between the table and the knowledge base, but it is unsupervised and it requires no external information. Interestingly, T2K relies on an iterative approach that, similarly to ours, alternates entity linking (i.e. matching HTML table rows to KB entities) and attribute matching (table columns to KB properties).

It is worth observing that, similarly to Sherlock and SATO, also in the table to knowledge base matching approaches, it is the case that semantic types are defined a priori (as they correspond to the types in the knowledge base), while in RAF- STD types arise from data. However, an unsupervised system like T2K can be adapted to discover semantic types by electing one source as the target knowledge base against which the remaining tables are matched.

In our experimental evaluation, we compare our solution to T2K, adapted to our context as described above.

Schema Matching The goal of schema matching, a widely studied topic in the last decades, is to find matches between semantically equivalent attributes [5,27,37]. Most schema matching approaches are able to deal with schematic heterogeneity across sources [21,26], such as differences in attribute names, domain formats and granularity. However, they assume that each source adopts homogeneous semantics and homogeneous representations of data. Coma++ [2] and Harmony [32] propose tools enabling a variety of matching approaches, configurable and adaptable to specific needs, and the reuse of previously defined strategies. However, these tools require some form of user interaction, such as selecting the most suitable techniques to adopt, or filtering all the possible matches suggested by the system. With a large number of sources, techniques that require humans in the loop are ineffective.

An interesting extension of Coma++ has been developed by Engmann and Massmann [19], who enhance the original schema matching system with an instance based-approach, i.e. leveraging also the extension of the datasets [37]. In particular, they augment Coma++ with two instance-based matchers, which compute a pairwise matching score among the attributes of the input datasets considering the presence of syntactical constraints and the similarity of values. The outcome of the matchers are then combined by a propagation algorithm [19]. According to the experimental evaluation recently conducted by Koutras *et al.* [27], this enhanced version of Coma++ outperforms other schema matching approaches.

The family of instance-based solutions to schema matching [37] includes other interesting proposals that aim at automatically performing attribute alignment by leveraging the attribute values. Autoplex [6] relies on a Bayesian classifier, whose training data must be provided as input, to match the attributes of a source to a target schema. A similar solution has been developed in the Big-Gorilla project, with FlexMatcher [12], an evolution of the LSD system [15]. Another instance based approach that is related to our work is Dumas [8], which infers and exploits partial record linkage information to discover alignment of schema attributes. These approaches are based on the assumption that, within each source, attributes have homogeneous values and semantics. Another schema matching approach that is worth mentioning is that proposed by Kang *et al.* [24], which is robust to differ-

ences in representation. However, it relies on homogeneity of the sources to get sufficient evidence of dependencies between attributes in the same source and requires *fully linked specifications*, while RAF- STD is able to work even with incomplete linkage.

Nguyen *et al.* [33], with their PSE system, address the problem of matching attributes in an existing catalog with attributes from external sources and do consider heterogeneity within sources. Having a complete catalog, they do not aim at discovering new attributes, but just at detecting those with an equivalent attribute in the catalog. Thanks to this restricted setting, they are more flexible to local heterogeneity in the sources. Indeed, if an attribute does not match with any catalog attribute (because it is not homogeneous, or it has too many errors, or it has a different format than the catalog), they simply neglect it. Indeed, their goal is not to be complete but to enrich their catalog, with specifications from Web sources. This approach is clearly a good compromise in their setting and can be easily adapted in our context by choosing a source as the initial catalog. Then, attributes that do not match with any of those of the catalog are added to the catalog itself, thus opening the possibility of a match with attributes of further sources.

In our experimental evaluation, we compare our solution to a version of PSE suitably adapted to our problem. Also, we have used the instance-based enhancement of Coma++ to evaluate the effectiveness of an internal module of the RAF-STD system.

3 Overview

We consider a set of sources $\mathbf{S}$ that provide information about entities of a specific category of interest (e.g. cameras or clothes). We assume each source $S \in \mathbf{S}$ to be a set of *records*, each of which is indeed an "entity *specification*" that describes the entity as a set of attribute name-value pairs. We represent an entity specification as a set of triples. Let us give some details, on notation and terminology. A *triple* has the form $\langle s, N, v\rangle$, where s identifies the entity specification, N is the *name* of an attribute, and v is the associated *value*. Since an attribute name appears at most once in an entity specification s, a triple is also briefly denoted as $s.N$ (with no mention of the associated value v). We denote as $S.N$ the set of triples in S whose name is N, and we call it a *source attribute* in S. Finally, we define the *domain of* $S.N$ as the set of all attribute values in its triples.

So, with respect to Fig. 1, $\langle s_{11}, \texttt{CPU}, \text{BionzX}\rangle$ is a triple, which can be simply denoted as $s_{11}.\texttt{CPU}$. Also, $S_1.\texttt{CPU}$ is a source attribute and its domain is {BionzX, Xpeed2}.

Our goal is to discover the *semantic type(s)* of the triples in $\mathbf{S}$, that is, to create clusters of triples that refer to the same semantic concept. Notice that we cannot associate semantic types to source attributes because of the heterogeneity that occurs even within the sources. Indeed, in two different specifications, even from the same source, information may be represented differently, and triples sharing the same attribute name may have different semantics. As a consequence, we need to work at the level of individual triples.

The definition of what is a semantic type might be subjective. We follow a *data-driven* approach, which considers semantic types as they emerge from the data offered by the sources. Referring to our running example, the presence of distinct triples that describe battery chemistry and battery model in the same specification (specification s_{11} in source S_1) suggests that these are distinct semantic types. Conversely, triples that provide values that appear as composite but with components that never emerge as distinct triples, are not considered separately. For example, the `Video resolution` triples also in S_1 could represent two distinct types (horizontal and vertical resolution), but as the two properties are never provided as separate triples, they are not (and should not be) considered separately.

We exploit the redundancy of information, which is naturally present on the Web with multiple specifications for the same entity. We assume, and take advantage of, the availability of a *good quality sample* of record linkage information, which identifies different specifications that refer to the same entity. Such a good quality sample need not be complete: in our experimental evaluation, in Sect. 6, we show that even a small sample suffices. Our assumption is motivated by the positive results recently achieved by several research projects that address the record linkage issue. In particular, systems based on pre-trained language models [10,28,42], as well as unsupervised systems [35,36,44], demonstrate that it is possible to obtain good quality record linkage information without attribute matching. Our approach has been designed to take advantage of such encouraging results.

Given a set of specifications $\mathcal{P}$, a *linkage sample* is a set $\mathbf{P}$ of disjoint subsets of $\mathcal{P}$, such that each $P \in \mathbf{P}$ contains only specifications that refer to the same entity (but not necessarily vice-versa). When two specifications s and t belong to the same element P of $\mathbf{P}$, we say they are *linked*. Similarly, we say that two triples are *linked* if the specifications to which they, respectively, pertain are linked.

3.1 Problem definition

Let us state the main topic of the paper, the *Fine Grained Semantic Type Discovery Problem*: given a set of sources $\mathbf{S}$ and a linkage sample $\mathbf{P}$ over the specifications in the sources in $\mathbf{S}$, find groups of triples from different entity specifications with the same semantic type. Groups may overlap, as individual triples may provide composite values, in which case

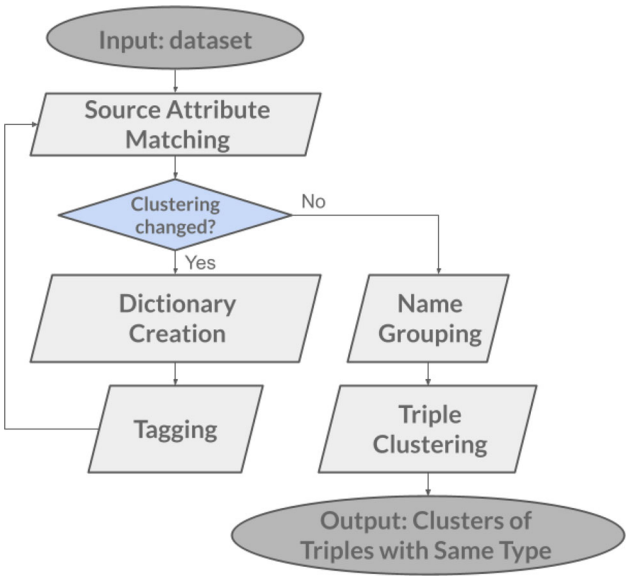

Fig. 3 The RAF- STD approach

they refer to several semantic types, associated with separate attributes in other specifications.

3.2 Our approach

To address the Fine Grained Semantic Type Discovery Problem, we exploit the opportunities that arise from the richness of information across sources. Our approach is illustrated in Fig. 3.

First, the *Source Attribute Matching* step exploits the existence of overlapping data across sources: we develop a Bayesian analysis that leverages the linkage sample to compute clusters of source attributes that share triples with matching values. Based on the intuition that it is unlikely that multiple linked triples have the same value by chance, this step aims at grouping together source attributes with the same semantics, while isolating, in singleton clusters, source attributes with overloaded names or value heterogeneity.

The non-singleton clusters built in the Source Attribute Matching step contain common values. The *Dictionary Creation* step exploits these values to build sets of values, dictionaries, that are likely to refer to the same semantic type.

Dictionary terms are then used by the *Tagging* step, to label values that belong to source attributes that were not clustered by the Source Attribute Matching, that is, source attributes that suffer attribute-name and attribute-value heterogeneity. Elements tagged with a given label may refer to the same semantic type of the triples in the cluster associated with such a label. We extract these values to create *virtual* attributes (and hence triples) that are added to the original dataset, each in the same specification as its original version.

Algorithm 1: Source Attribute Matching

Input : **S**: set of sources
C: initial set of clusters over $atts(\mathbf{S})$
P: linkage sample over **S**
Output: a set **C** of clusters over $atts(\mathbf{S})$

1 Construct a graph $G(V, E)$ as follows: $V = atts(\mathbf{S})$, E contains a weighted edge $e_{A,B}$, with $weight(e) = \text{SIM-SCORE}(A, B, \mathbf{P})$, between a pair of source attributes A and B, $A \in S_i$, $B \in S_j$, $S_i \neq S_j$, if A and B share a value in at least one linked specification and $\text{SIM-SCORE}(\cdot) \geq 0.5$;
2 **for** *each edge* $e_{A,B} \in E$ *ordered by desc weight* **do**
3 $c_A \leftarrow c \in \mathbf{C} : A \in c$;
4 $c_B \leftarrow c \in \mathbf{C} : B \in c$;
5 **if** $\nexists s : X \in s, Y \in s, X \in c_A, Y \in c_B$ **then**
6 $\mathbf{C} \leftarrow (\mathbf{C} \setminus \{c_A, c_B\}) \cup \{c_A \cup c_B\}$; // merge
7 **end**
8 **end**
9 **return C**;

Then, the process iterates: virtual attributes generated by the Tagging step can give rise to new matches that could not have been identified previously.

As every Source Attribute Matching step can produce new clusters (and then new dictionaries), and every Tagging step can generate new virtual attributes that allow new matching, the two steps are iterated until the clusters do not change anymore.

After the end of iterations, the *Name Grouping* step aggregates clusters having source attributes with common names and comparable domains. This final step also exploits the redundancy of data, as the domain of clusters is now more reliable than domains of single source attributes and can match source attributes that did not match previously, typically for lack of linkage, excessive noise in the values, or too generic values.

At the final stage, *Triple Clustering* creates clusters by aggregating triples that belong to the same cluster of (possibly virtual) source attributes.

4 Source attribute matching

In this section, we describe the first step of our approach, which leverages the linkage sample and redundancy of data among sources to identify groups of source attributes with equivalent semantics. This is performed by Algorithm 1, which takes as input an initial clustering of source attributes and refines it by potentially merging some clusters. Notice that the algorithm is applied iteratively (see Sect. 5.2) and that in the first iteration all clusters are initialized as singletons.

Let $atts(\mathbf{S})$ be the set of all the source attributes in **S**. We build a weighted graph, $G(atts(\mathbf{S}), \mathbf{E})$, whose nodes are source attributes in $atts(\mathbf{S})$ and edges are candidate matches, i.e. pairs of source attributes from different sources with at

least one common value in a pair of linked specifications, excluding values that are too frequent and thus not very meaningful. [3] Each edge has a weight given by the scoring function SIM-SCORE, based on a Bayesian analysis and described in Sect. 4.1, which gives the probability of match between pairs of source attributes. Edges with match probability smaller than non-match probability (that is, SIM-SCORE < 0.5) are dropped.

Then, edges are sorted and processed in descending order of weight. For each edge, the two clusters it connects are merged, unless this would put two triples from the same specification in the same cluster (we assume that two triples with same semantic type but conflicting values cannot appear in the same specification). As we detail in Sect. 4.2, we use approximate matching to compare values.

4.1 Similarity score

The similarity score between two source attributes is computed by means of a probabilistic Bayesian approach that leverages the linkage sample. The intuition is that if two source attributes share the same values for several triples, then it is likely that they represent the same property. The approach aims to be tolerant to the presence of noise in the values as well as in the linkage.

Given two source attributes $A = S_i.N$, $B = S_j.M$, with $S_i \neq S_j$, we use L_{AB} to denote the set of pairs $\langle s.N, t.M\rangle$ where $s \in S_i$, $t \in T_j$, s and t are linked.

Our goal is to determine $P(A \equiv B|L_{AB})$, that is, the probability that source attributes A and B are equivalent (that is, they have the same meaning), given the pairs of the values of source attributes A and B in the linked specifications.

By applying Bayes' theorem, we have:

$$P(A \equiv B|L_{AB}) = \frac{P(A \equiv B)P(L_{AB}|A \equiv B)}{P(L_{AB}|A \equiv B)P(A \equiv B) + P(L_{AB}|A \not\equiv B)(1 - P(A \equiv B))} \quad (1)$$

Let us illustrate next the development of this formula.

4.1.1 Prior probability

Given a pair of source attributes A and B, we estimate the prior probability $P(A \equiv B)$ by heuristically considering the similarity of their domains, denoted $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively. In particular, we take into account (i) the similarity of the whole domains, and (ii) the similarity of the domains restricted to the values shared by the linked specifications.

We compute the similarity of the domains according to the Chekanovsky–Sørensen index [9,40], a variant of the Jaccard index that considers the distribution of values in the sets. Also, we weigh the contribution of each value by its frequency in the set of source attributes; the rationale is that sharing values that are very frequent in the dataset is less informative than sharing rare values. Formally:

$$sim(A, B) = \sum_{v \in \mathcal{D}_A \cap \mathcal{D}_B} 2\frac{min(f(v, \mathcal{D}_A), f(v, \mathcal{D}_B))}{|\{Y \in atts(:)v \in \mathcal{D}_Y\}|} \quad (2)$$

where $f(v, \cdot)$ is the frequency of v in $\cdot$, and the denominator $|\{Y \in atts(:)v \in \mathcal{D}_Y\}|$ is the number of source attributes in which the value v appears.[4]

We use the same approach to compute the similarity of the domains restricted to the values from the linked specifications (we call this *linked domain*):

$$sim_L(A, B) = \sum_{\langle v,v\rangle \in L_{AB}} 2\frac{min(f(v, L_A), f(v, L_B))}{|\{Y \in atts(:)v \in \mathcal{D}_Y\}|} \quad (3)$$

where L_A (and similarly L_B) is the set of values of A in L_{AB}, i.e. $L_A = \{v : v$ is the value in $s.A$ and $\langle s.A, t.B\rangle \in L_{AB}\}$.

In order to compute the prior probability $P(A \equiv B)$, we mostly rely on values that are provided by linked specifications. However, if there are too few elements in linkage, values could be shared by coincidence, and thus, we should give more weight to the similarity between the entire domains. To weigh the two components, we consider their sizes; however, since they can differ by many orders of magnitude, we use their logarithm, as follows:

$$P(A \equiv B) = \frac{\alpha\, sim(A, B) + \beta\, sim_L(A, B)}{\alpha + \beta} \quad (4)$$

where $\alpha = log(\frac{max(|\mathcal{D}_A|,|\mathcal{D}_B|)}{|L_{AB}|})$ and $\beta = log(|L_{AB}|)$. Note that if $|L_{AB}| = 1$ (there is just one pair of linked specifications) β equals 0, then we consider only the similarity of the domain. Conversely, increasing $|L_{AB}|$ reduces the weight of α and thus gives more importance to the values coming from the linked specifications.

Example 1 Consider the source attributes S_3.`Memory` and S_4.`Memory` from our running example in Figure 1. For the sake of conciseness, we use A and B to denote S_3.`Memory` and S_4.`Memory`, respectively. Table 1 reports the frequencies of the values in the domains associated with the two source attributes (f_A and f_B); their frequencies in the domains restricted to the linked specifications (f_{L_A} and f_{L_B}); the number of source attributes in which v occurs ($occ(v) = |\{S.Y, S \in \mathbf{S} : v \in \mathcal{D}_{S.Y}\}|$).

[3] We exclude values that appear in the domain of more than 10% of source attributes in the dataset.

[4] The numerator includes 2 because by construction we are considering values that appear in the intersection of two sources.

Table 1 Domain of source attributes $S3.\texttt{Memory}$ (denoted A) and $S4.\texttt{Memory}$ (denoted B)

v	f_A	f_B	f_{L_A}	f_{L_B}	$occ(v)$
16	0.5	0.66	0.33	0.66	3
8	0.25	0.33	0.33	0.33	3
32	0.25	0	0.33	0	1

By applying Equations 2 and 3 , $sim(A, B) = 0.5$ and $sim_L(A, B) = 0.44$. The weights assume the values: $\alpha = 0.42$, $\beta = 1.58$. Then, from Equation 4, we obtain $P(A \equiv B) \approx 0.45$. In a similar way, we can compute the prior for source attributes $S_4.\texttt{Memory}$ and $S_5.\texttt{Megapixels}$, denoted B and C, respectively: $P(B \equiv C) = 0.64$. These two source attributes have very similar domains, thus a high prior, despite providing different data. Example 2 will show how we exploit linkage to adjust the score.

4.1.2 Posterior probability

We now need to compute the probability of observing the values provided for each specific pair of triples in linkage:

- under the null hypothesis, $P(L_{AB}|A \not\equiv B)$,
- under the equivalence hypothesis, $P(L_{AB}|A \equiv B)$.

We can model the set of the observed value pairs L_{AB} as a set of independent events, one for each linked pair of specifications:[5]

$$P(L_{AB}|\cdot) = \prod_{\langle s.A, t.B\rangle_i \in L_{AB}} P(s.A, t.B|\cdot) \tag{5}$$

Null hypothesis Under the null hypothesis, the two source attributes are not aligned and we can assume that they are independent:

$$P(s.A, t.B|A \not\equiv B) = P(s.A|A \not\equiv B)P(t.B|A \not\equiv B).$$

We model the value provided by each attribute in a triple as the outcome of a random variable, where the probability of each value is estimated with its frequency in the domain of the corresponding attribute. Therefore,

$$P(s.A, t.B|A \not\equiv B) = f(s.A, \mathcal{D}_A) f(t.B, \mathcal{D}_B). \tag{6}$$

[5] The independence assumption is indeed a simplification, and the individual probability estimation of each observation may not be very precise. However, as with Naive Bayes, given the number of evidence, decisions based on the final probability score are reliable [30], as our experiments show.

Equivalence hypothesis Under the equivalence hypothesis, the two source attributes represent the same real-world property, which we model by means of a random variable X.

In order to compute $P(s.A, t.B|A \equiv B)$, we need to distinguish two cases: the values provided by the two source attributes are either (i) different or (ii) equal. Intuitively, we expect that they are equal (we are under the equivalence hypothesis), unless one of the two values (or both) is wrong.

Let us consider first the case in which $s.A$ and $t.B$ are different. Either only one of them provides the actual value of X, or they are both wrong and so none provides the actual value of X:

$$\begin{aligned} &P(s.A = v_1, t.B = v_2, v_1 \neq v_2|A \equiv B) = \\ &\quad P(X = s.A = v_1, t.B = v_2, v_1 \neq v_2|A \equiv B) + \\ &\quad P(s.A = v_1, X = t.B = v_2, v_1 \neq v_2|A \equiv B) + \\ &\quad P(s.A = v_1, t.B = v_2, X \notin \{v_1, v_2\}|A \equiv B) \end{aligned}$$

By applying the conditional probability definition:

$$\begin{aligned} &P(s.A = v_1, t.B = v_2, v_1 \neq v_2|A \equiv B) = \\ &\quad P(X = v_1|A \equiv B)P(s.A = v_1|X = v_1, A \equiv B) \\ &\quad\quad P(t.B = v_2|X = v_1, A \equiv B) + \\ &\quad P(X = v_2|A \equiv B)P(s.B = v_1|X = v_2, A \equiv B) \\ &\quad\quad P(t.A = v_2|X = v_2, A \equiv B) + \\ &\quad P(s.A = v_1|X \notin \{v_1, v_2\}, A \equiv B) \\ &\quad\quad P(t.B = v_2|X \notin \{v_1, v_2\}, A \equiv B) \\ &\quad\quad P(X \notin \{v_1, v_2\}|A \equiv B) \end{aligned} \tag{7}$$

Since we are under the hypothesis that the source attributes A and B are equivalent, the union of their domains represents an approximation of the domain of X. Then, we can estimate $P(X = v|A \equiv B)$ considering its frequency in $\mathcal{D}_A \cup \mathcal{D}_B$:

$$P(X = v|A \equiv B) = f(v, \mathcal{D}_A \cup \mathcal{D}_B) \tag{8}$$

To compute $P(s.A = v'|X = v, A \equiv B)$, we need to distinguish whether $t.A$ is correct (it equals the value of the random variable X) or wrong.

An attribute can provide a wrong value because of an error in the source or because of a linkage error. In our model, we can consider linkage errors as a special case of source errors. Therefore, we assume that each attribute has the same error probability ϵ for every observation and that, in case of error, the attribute provides a random value from those of the domain of the property, which is estimated by $\mathcal{D}_A \cup \mathcal{D}_B$:

$$\begin{aligned} &P(s.A = v'|A \equiv B, X = v) = \\ &\begin{cases} 1 - \epsilon + \epsilon f(v, \mathcal{D}_A \cup \mathcal{D}_B) & [v' = v] \\ \epsilon f(v, \mathcal{D}_A \cup \mathcal{D}_B) & [v' \neq v] \end{cases} \end{aligned} \tag{9}$$

Notice the $+\epsilon f(v, \mathcal{D}_A \cup \mathcal{D}_B)$ term in the first row: the random value provided in case of error may be the correct value by chance. This assumption models the fact that frequent values are less likely to be mistaken.

By replacing Equations 8 and 9 in Equation 7:

$$P(s.A = v_1, t.B = v_2, v_1 \neq v_2 | A \equiv B) = \epsilon(2-\epsilon) f(v_1, \mathcal{D}_A \cup \mathcal{D}_B) f(v_2, \mathcal{D}_A \cup \mathcal{D}_B) \quad (10)$$

Notice that, if $\epsilon = 0$ (perfect sources on these source attributes), the above probability equals 0, i.e. triples with the same semantics from two linked specifications cannot provide different values.

Let us now consider the case in which $s.A$ equals $t.B$. Either they are correctly providing the actual value of X (the sources are not making errors), or both of them are wrong and they are assuming the same value by chance.

$$\begin{aligned} &P(s.A = t.B = v | A \equiv B) = \\ &\quad P(s.A = t.B = X = v | A \equiv B) + \\ &\quad P(s.A = t.B = v, X \neq v | A \equiv B) \end{aligned} \quad (11)$$

By the conditional probability definition, the probability of the first term is given by the probability that the correct value for the semantic type described by the source attributes is v multiplied by the probabilities that A and B are correctly providing v, that is: $P(X = v | A \equiv B) P(s.A = v | X = v, A \equiv B) P(t.B = v | X = v, A \equiv B)$. Similarly, the second term results: $P(X \neq v | A \equiv B) P(s.A = v | X \neq v, A \equiv B) P(t.B = v | X \neq v, A \equiv B)$.

By applying Eq. 8 and 9 , we obtain:

$$\begin{aligned} &P(s.A = t.B = v | A \equiv B) = \\ &f(v, \mathcal{D}_A \cup \mathcal{D}_B)(1 - \epsilon(2-\epsilon)(1 - f(v, \mathcal{D}_A \cup \mathcal{D}_B)) \end{aligned} \quad (12)$$

If the sources are perfect ($\epsilon = 0$), the above probability equals $f(v, \mathcal{D}_A \cup \mathcal{D}_B)$, i.e. it corresponds to the frequency of the value (which is estimated by the union of the domains).

Summary The probability $P(A \equiv B | L_{AB})$ that two source attributes A and B are equivalent, given the pairs of the values of source attributes A and B in the linked specifications, is obtained by replacing in Equation 1 the prior probability, $P(A \equiv B)$, and the posterior probabilities under the null, $P(L_{AB} | A \not\equiv B)$, and the equivalence hypothesis, $P(L_{AB} | A \equiv B)$. The prior probability is computed by Equation 4. The posterior probabilities are computed by using Equation 6 (null hypothesis) and Equation 12 (equivalence hypothesis) as factors in the multiplication of Equation 5.

Example 2 Continuing Example 1, after computing probability of equivalence under null and equivalence hypothesis for source attributes S_3.`Memory`, S_4.`Memory` (denoted A and B, respectively) and S_4.`Memory`, S_5.`Megapixels` (denoted B and C), we obtain with Equation 1 the final scores: $P(A \equiv B | L_{AB}) \approx 0.86$, and $P(B \equiv C | L_{BC}) \approx 0.48$. Even though the prior probability $P(B \equiv C) = 0.64$ was larger than the prior probability $P(A \equiv B) = 0.45$, the posterior probabilities are flipped, reflecting the evidence. This example shows how the Bayesian model exploits linkage information to distinguish attributes that may have very similar domains.

4.2 Approximate match

It turns out that, in comparing values, an approximation is often needed. Therefore, it is important to be tolerant in comparisons, and so we pre-process values as follows:[6] *(i)* tokens are split at each number–letter, letter–number, lowercase–uppercase transition; *(ii)* sequences of non-alphanumeric characters are replaced with a single whitespace, except for commas and dots in numeric sequences; *(iii)* accents and diacritics are removed; *(iv)* uppercase letters are converted to lowercase; *(v)* values are converted to a set of tokens (e.g. "12MP frontCamera-11.5MP rearCamera" → {12, mp, front, camera, 11.5, rear}).

While computing the similarity score (Sect. 4.1), we consider two values as equivalent if the Jaccard similarity between their tokens is greater than a given threshold.[7]

5 Dictionary creation and tagging

The second step of our approach leverages the Source Attribute Matching results to resolve issues related to attribute-name and attribute-value heterogeneity.

For each non-singleton cluster computed by the Source Attribute Matching step, we derive a *dictionary* of values given by the union of the domains of its source attributes. Such a dictionary is an approximation of the domain of the semantic type represented by the cluster. Then, in every attribute value we tag the strings that match a term of a dictionary. Each tagged string gives rise to a *virtual* attribute.

Observe that the creation of virtual attributes is a step toward the solution of a value heterogeneity that prevented the match with other source attributes. In our example, S_5.`Batt` cannot match to S_1.`Battery chemistry` because of the strings that decorate the values of S_5.`Batt` ("included", "battery", "rechargeable"). Analogously, S_3.`Battery` can match neither S_1.`Battery chemistry` nor S_1.`Battery model` because its values are composite (they contain both the battery chemistry and the battery model).

[6] Notice that this normalization is not applied in the tagging step, described in Section 5.1.

[7] We have set the threshold at 0.9.

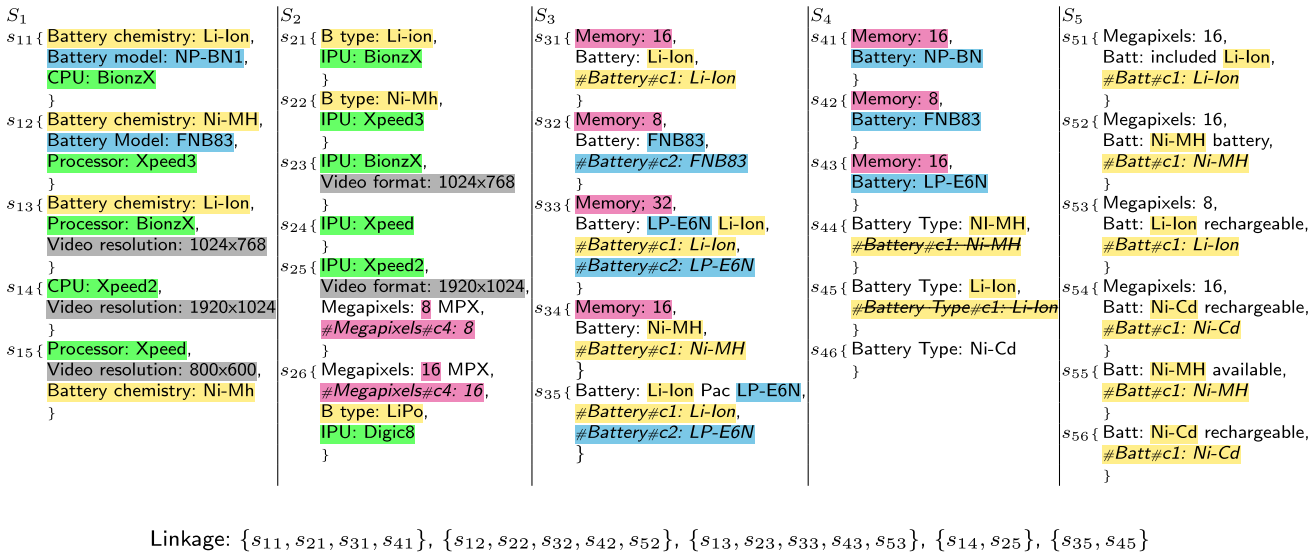

Fig. 4 Tagged values and virtual attributes in the sources of the running example (original sources are in Fig. 1). Background colors denote clustering of triples. Virtual attributes are in italics and are marked by a hash symbol ("#"): they were not in the original specifications and have been added by the first iteration of the algorithm. Strikethrough virtual attributes (those in $S4$) will be deleted in subsequent iterations

The virtual attributes created by the tagging step allow to isolate the values and possibly to trigger new matches.

The above observations motivate our iterative approach: virtual source attributes can produce larger clusters, which can in turn enrich dictionaries and create new virtual source attributes.

Let us now illustrate the details of these steps.

5.1 Tagging and virtual attributes generation

The first phase for the extraction of virtual attributes consists of creating dictionaries of values for the clusters of source attributes obtained by the Source Attribute Matching step. We associate each non-singleton cluster $c \in \{c \in \mathbf{C}, |c| > 1\}$ with a *dictionary*, denoted $\mathcal{D}_c$, containing the union of the domains of its source attributes: $\mathcal{D}_c = \bigcup_{A \in c} \mathcal{D}_A$. For the sake of efficiency, we exclude very long values, as they usually correspond to noisy or mixed attributes. Also, we filter out values that are present in many clusters, as they do not characterize the domain. Typically these values have generic meanings (such as, "Yes", "No", "Not available") that can apply to many source attributes, even with completely different semantics.[8]

We associate a label l_c with each cluster c, and we tag with l_c every string (sequence of tokens) contained in any attribute value $\mathtt{A} \notin c$ that matches a term in $\mathcal{D}_c$. If a source attribute $S.A$ contains at least two attribute values that have been tagged with the same label l_c, then we extract the tagged strings and use them as values for a new *virtual attribute*, whose name is denoted `#A#c`.

It is worth observing that a given value could be tagged with many labels (because terms from different dictionaries match with different portions of the value). Whenever a tagged string is contained in another tagged string, we consider as more reliable the match with the larger term, and hence, we drop the label of the smaller tagged string.

Example 3 In our running example of Figure 1 after the first matching step, we have the following non-singleton clusters:
$c_1 = \{S_1.\texttt{Battery chemistry}, S_2.\texttt{Btype}\}$
$c_2 = \{S_1.\texttt{Battery model}, S_4.\texttt{Battery}\}$
$c_3 = \{S_1.\texttt{CPU}, S2.\texttt{IPU}, S_1.\texttt{Processor}\}$
$c_4 = \{S_3.\texttt{Memory}, S_4.\texttt{Memory}\}$
$c_5 = \{S_1.\texttt{Video Resolution}, S_2.\texttt{Video Format}\}$
whose dictionaries result as follows:

$\mathcal{D}_{c_1} = \{$Li-Ion, Ni-MH, LiPo$\}$
$\mathcal{D}_{c_2} = \{$NP-BN1, FNB83$\}$
$\mathcal{D}_{c_3} = \{$BionzX, Xpeed3, Xpeed2, Xpeed, Digic 8$\}$
$\mathcal{D}_{c_4} = \{8, 16, 32\}$
$\mathcal{D}_{c_5} = \{$1024x768, 1920x2014, 800x600$\}$

Figure 4 shows the sources enriched by the values tagged with the dictionaries' terms and the virtual attributes created accordingly. Several values of the source attribute $S_3.\texttt{Battery}$ are tagged with terms in $\mathcal{D}_{c_1}$ and $\mathcal{D}_{c_2}$, leading to the creation of virtual source attributes `#Battery#c1` and `#Battery#c2`, and hence identifying the homonym of the source attribute $S_3.\texttt{Battery}$. Tagging values of $S_5.\texttt{Batt}$, which suffers from representation heterogeneity,

[8] Specifically, we drop values with more than 25 characters and values that are present in more than 10% of the clusters.

with terms in $\mathcal{D}_{c_1}$ allow the creation of the virtual source attribute #Batt#c1, whose values are cleansed from uninformative pieces of text. No virtual source attribute is created with elements of cluster c_4 tagged in S_5.Megapixels since it would be identical to the original source attribute and thus useless. A virtual source attribute is extracted from S_2.Megapixels using terms in $\mathcal{D}_{c_4}$. This virtual source attribute will not match any other attribute in the subsequent source attribute matching step and thus will remain isolated. Notice that the source attribute S_4.Battery Type was not part of cluster c_1 as it does not have enough linkage for matching. However, its values in s_{44} and s_{45} were tagged with terms from $\mathcal{D}_{c_1}$, giving rise to the virtual source attribute #Battery Type#c1.

5.2 Iterating matching and tagging

Matching and tagging are launched iteratively, as one provides new evidence that can be exploited by the other.

Algorithm 2 presents the pseudo-code of the overall procedure. Let us comment on the various steps. Source attribute clustering **C** is initialized with a singleton cluster for each source attribute. The first execution of the matching step (line 2) produces a new version of the clustering.

Then, the iteration starts (line 3). The current clustering along with the full dataset is provided as input to the dictionary creation step (line 4), which returns a dictionary for each cluster. The dictionaries are exploited by the tagging step that produces a new version of the dataset, $\mathbf{S}'$, introducing virtual source attributes, as described in Sect. 5.1.

Each iteration takes as input the clusters created in the previous iteration, excluding virtual source attributes (line 5), which are re-generated at each new iteration. These steps are repeated until the set of clusters **C** does not change any longer (line 8). [9] The algorithm converges, as the matching step never splits clusters, but it can only potentially merge them.

It is important to observe that the tagging step (line 6) is always done on the original version of dataset, **S**, which does not include any virtual source attribute, but using the latest version of the dictionaries, which are created after each matching step. In this way, at every iteration the dictionaries accumulate knowledge about the domain of each cluster; the tagging step takes advantage of the enhanced dictionaries to tag more values, giving rise to more accurate virtual source attributes, which trigger new source attribute matches.

[9] Notice that we compare clustering only of non-virtual source attributes. Indeed, if an iteration adds only virtual source attributes, then the subsequent iteration would have no effect: virtual source attributes do not contribute to dictionaries, so dictionary and clustering would not change.

Algorithm 2: Matching and tagging iteration

Input : **S**: set of sources
Output: $\mathbf{S}'$: **S** augmented with virtual source attributes
C: set of source attribute clusters over $atts(\mathbf{S}')$)

1 $\mathbf{C} \leftarrow \{\{A\}, A \in atts(\mathbf{S})\}$; `// singleton clusters`
2 $\mathbf{C} \leftarrow sourceAttributeMatching(\mathbf{S}, \mathbf{C})$;
3 **do**
4 $\quad \mathbf{D} \leftarrow \{\mathcal{D}_c, c \in \mathbf{C}, |c| > 1\}$;
5 $\quad \mathbf{C}_{prev} \leftarrow excludeVirtual(\mathbf{C})$;
6 $\quad \mathbf{S}' \leftarrow tag(\mathbf{D}, \mathbf{S})$; `// add virtual attr.s to` **S**
7 $\quad \mathbf{C} \leftarrow sourceAttributeMatching(\mathbf{S}', \mathbf{C}_{prev})$;
8 **while** $excludeVirtual(\mathbf{C}) \neq \mathbf{C}_{prev}$;
9 **return C**;

Example 4 To illustrate the interaction between tagging and matching, let us continue Example 3, where we showed the results of the first tagging step, which created the virtual source attributes shown in Figure 1. In the subsequent matching step source attributes S_3.#Battery#c1, S_4.Battery Type, S_4.#Battery Type#c1 and S_5.#Batt#c1 are added to cluster c_1, while S_3.#Battery#c2 is added to cluster c_2. Notice the source attribute S_4.Battery Type, which remained isolated in the previous matching step because of lack of linkage (s_{35} and s_{45} are in linkage, but the values of s_{35}.Battery and s_{45}.Battery Type do not match, because of the mixed values in s_{35}.Battery). S_4.Battery Type has now enough linkage and matching with the virtual source attribute S_3.#Battery#c1, and therefore, it is included in the same cluster. Observe that the presence of S_4.BatteryType contributes improving the dictionary associated with c_1 with an additional value, "Ni-Cd". This has a positive impact in the successive tagging step (which occurs in the next iteration) as the dictionary enhanced with such a value allows the creation of two new virtual attributes (in grey with white lines), $\langle s_{54}$, #Batt#c1, Ni-Cd$\rangle$ and $\langle s_{56}$, #Batt#c1, Ni-Cd$\rangle$, which enrich the virtual source attribute s_{56}.#Batt#c1. With the new dictionary, source attribute S_4.#Battery#c1 (strikethrough) is not created anymore, as it would be identical to S_4.Battery.

5.3 Name Grouping

Before building the final clusters of triples, we perform the *Name Grouping Step*, which aims at merging the clusters obtained by the iterations based on features related to global homogeneity that occur among the sources.

Observe that the lack of linkage, or strong forms of representational heterogeneity could prevent the creation of some clusters. However, at this stage, since the iterative approach solved several heterogeneity issues, we can exploit classical schema matching strategies, based on attribute names and domain similarity, to further group the generated clusters.

Cluster pairs that have at least a pair of source attributes with the same name are selected as candidates for merging. We prevent to select candidates from isolated source attributes from which at least a virtual attribute has been generated and clustered in the subsequent source attribute matching step. Indeed, this is a signal of a possible presence of attribute name or attribute value heterogeneity, and thus, its semantics may be ambiguous.

A candidate pair of clusters $\{c_a, c_b\}$ are merged if their values overlap. In order to evaluate the overlap of values, we consider the maximum Jaccard Containment [22] between the sets of tokens in the values of the source attributes in the two clusters. The use of tokens, instead of values, allows merging of attributes with noisy values. Let T denote the set of the tokens in the values of the source attributes of a cluster c. A pair of candidate clusters $\{c_a, c_b\}$ is merged if $\frac{|T_a \cap T_b|}{min(|T_a|,|T_b|)}$ is greater than a threshold.[10]

Example 5 The source attributes S_2.`Megapixels` and S_5.`Megapixels`, which are isolated, are selected for merging. Their token sets are {8, 16, MPX} and {8, 16}, respectively. Tokens from the second set are all contained in the first one, so their similarity is 1, and the two attributes are merged to form a single cluster. Note that S_2.`Megapixels` produced only a virtual attribute that remained isolated in the subsequent source attribute matching step. The source attributes S_3.`Battery` and S_4.`Battery` are not selected for merging because S_3.`Battery` generated two virtual attributes that were clustered in subsequent source attribute matching step.

5.4 Triple clustering

The final step of our approach consists in creating clusters of triples, which represent the semantic types identified by our approach.

For each non-singleton cluster $c \in \mathbf{C}$, we create a new triple cluster, which we denote ST (semantic type). As the goal of RAF- STD is to detect matching between attribute instances and not to extract the specific value, each cluster of triples contains all the triples of the non-virtual source attributes in c and all the original triples of its virtual source attributes.

Example 6 The algorithm iterates over each non-singleton cluster in $\mathbf{C}$ to build a corresponding triple cluster. Consider cluster c_1: source attributes S_1.`BatteryChemistry`, S_2.`Btype` and S_4.`BatteryType` are not virtual, so we simply add their triples to the result. Source attributes S_3.`#Battery#c1` and S_5.`#Batt#c1` are virtual, so for each of their triples we add to the cluster their corresponding original triple. Indeed, we will not add s_{32}.`Battery`, containing only the model of the battery. The final cluster of triples would be: ST_1 = {s_{11}.`Battery Chemistry`, s_{12}.`Battery Chemistry`, s_{13}.`Battery Chemistry`, s_{21}.`B Type`, s_{22}.`BType`, s_{26}.`BType`, s_{31}.`Battery`, s_{33}.`Battery`, s_{34}.`Battery`, s_{35}.`Battery`, s_{44}.`Battery Type`, s_{45}.`Battery Type`, s_{46}.`Battery Type`, s_{51}.`Batt`, s_{52}.`Batt`, s_{53}.`Batt`, s_{54}.`Batt`, s_{55}.`Batt`, s_{56}.`Batt`}. Figure 2 shows a portion of the above results.

5.5 Complexity analysis

Let N be the total number of source attributes, and K the average number of triples per source attribute ($N \cdot K$ = total number of triples). In the worst case, Algorithm 2 performs N iterations (all the source attributes are added to a cluster, one source attribute per iteration). [11] Each iteration is composed of: (i) the dictionary creation step, whose cost is linear in the number of triples that belong to a non-isolated cluster: in the worst case $O(N \cdot K)$; (ii) the tagging step, which inspects sub-strings[12] of each attribute value, to match dictionary entries. Let T be the average number of tokens of each attribute value: the cost of the tagging step is $O(N \cdot K \cdot T)$; (iii) the Source Attribute Matching step.

This last step (Algorithm 1) performs in turn three operations:

- First, it detects pairs of source attributes with a common value for entities in linkage. This is done efficiently by means of an inverted index built on each group of specifications in linkage. It associates each value with all source attributes that provide such a value in this group. Then, its cost is $O(N \cdot K)$.
- Then, for each selected pair, it computes the matching score (SIM- SCORE). This operation requires to compute prior and posterior probabilities, which need to compare, respectively, source attribute domains and values for pairs of linked triples. In both cases, the cost is linear in the minimum size of domains of the two source attributes. In the worst case, all source attributes share at least one common value for all the specifications in linkage (i.e. all possible pairs of source attributes are selected), and all source attributes have the same size (K). Thus, the cost of this step is $O(N^2K)$.
- Finally, source attribute pairs with scores higher than 0.5 are sorted by decreasing weight, with a cost of $O(N^2 log(N^2))$.[13] In practice, the number of selected

[10] We have empirically set the value of such a threshold 0.8.

[11] In our experiments, we observe at most 5 iterations.

[12] Up to 4 tokens.

[13] The complexity remains the same in all iterating steps, as we limit the number of virtual source attributes per original source attribute to a constant, 50 (in our experiment we observe that at most the number of source attributes is doubled).

Table 2 Details on the evaluation datasets

	DI2KG		WDC		
	Camera	Monitor	Clothing	Jewelry	Auto
Sources	24	26	185	379	534
Specifications	29,787	16,662	14,618	27,039	47,112
Triples	528,186	443,564	136,609	235,192	426,563
Source Attributes	6,560	3,711	1,970	5,616	8,343
Entities in linkage	100	196	759	3,362	6,892
Specifications in linkage	2,826	2,070	6,798	11,937	22,117
Linked specification pairs	56,503	9,087	74,580	27,442	23,947
Ground-truth clusters	56	83	93	168	41
Source attributes in the ground truth	687	1,026	851	1,536	1,597

pairs is much lower, and the number of edges with a score higher than 0.5 is even less.[14]

The final cost of the algorithm in the worst case is then $N * [O(N \cdot K) + O(N \cdot K \cdot T) + O(N \cdot K) + O(N^2K) + O(N^2 log(N))] = O(N^2 \cdot K \cdot T + N^3 \cdot K + N^3 \cdot log(N))$. Name grouping ($O(N^2 \cdot K \cdot T)$) and triple clustering ($O(N \cdot K)$) do not change final cost.

6 Experiments

In this section, we report the experimental evaluation of RAF- STD for semantic type discovery.[15]

In Sect. 6.1, we analyse the impact of each step of our approach, its robustness and the role of linkage. Then, in Sect. 6.2, we compare RAF- STD to several alternative approaches. For these experiments, we use datasets from the DI2KG benchmark. In Sect. 6.3, we extend our comparison to alternative approaches running experiments on three datasets from the *WDC Product Data Corpus (WDC)* [35]. Finally, in Sect. 6.4 we presents results about the scalability of the approach.

Table 2 summarizes the characteristics of the DI2KG and WDC datasets that are used in our evaluation. For both the datasets, the ground truth consists of a set of clusters, each containing triples related to a given semantic type (e.g, battery chemistry). Because of the presence of composite attributes, clusters may overlap. The ground-truth datasets are not guaranteed to be complete: each cluster contains *only* correct triples, but not necessarily *all* of them.[16] More details about the datasets and their associated ground truth are described in the following sections.

Metrics In order to evaluate our approach, we consider precision, recall and F-measure (their harmonic mean) of our clusters of triples with respect to the ground-truth clusters. It is important to remind that clusters may overlap, as some triples may contain composite values that refer to different semantic types. For example, in Fig. 2, clusters ST1 and ST2 share one triple ($\langle s_{33}$, `Battery`, LP-E6N Li-Ion$\rangle$) as its value refers to both the battery chemistry (ST1) and the battery type (ST2). Therefore, we evaluate results over *pairs of triples* that match, i.e. pairs of triples that share *at least* one cluster [3]. Precision is computed as usual as the fraction of correct pairs over all the pairs in match the algorithm provided; recall as the fraction of actual pairs in match that the algorithm found. Because of the incompleteness of the ground truth, evaluation is limited to all the pairs that occur in the ground-truth clusters: if the algorithm provides two triples in the same cluster, and one or both are not in the ground truth, this pair is ignored and not considered as a true or false positive. However, if both triples are in the ground truth, but they do not belong to the same cluster (they do not form a pair), the case is considered as a false positive.

6.1 Evaluation of RAF- STD

Our primary benchmark to evaluate RAF- STD is DI2KG, which provides two datasets, Camera (24 sources) and Monitor (26 sources).

The two DI2KG datasets are associated with a manually curated ground truth for semantic type discovery, and with a golden linkage sample. [17] The semantic type discovery ground truth consists of 56 (Camera) and 83 (Monitor) clusters, originated from 687 (Camera) and 1,026 (Monitor) source attributes. The golden linkage sample corresponds

[14] For example, in the DI2KG camera dataset there are 985 pairs in the first iteration (415 with score > 0.5) and 44,560 in the second iteration (2,950 with score > 0.5).

[15] Source code and datasets are available at: https://github.com/merialdo/research.raf.

[16] The datasets are too large, and manually producing the ground truth is unfeasible.

[17] In the DI2KG website, they are called instance-level attribute matching and entity resolution, respectively.

Table 3 Results on algorithm steps

	Match			Match + Tag			Full algorithm		
	P	R	F1	P	R	F1	P	R	F1
Camera	**0.99**	0.5	0.66	0.98	0.65	0.78	0.97	**0.76**	**0.85**
Monitor	**0.99**	0.32	0.48	0.94	0.56	0.70	0.93	**0.65**	**0.76**

to 14% of the total linkage information of the dataset. It has been created by running several entity matching systems on the whole dataset and merging their results. A sample of the obtained groups of entities has been manually curated, in order to create a complete and clean golden set. The sample for the Camera dataset is composed of 100 groups of entities, containing a total of 2,826 specifications, with 56,503 linked specification pairs. For Monitor, there are 196 entity groups, involving 2,070 specifications and 9,087 specification pairs. The sampling has been conducted paying attention to select clusters that were representative of the distribution of the groups size in the dataset (considering the distribution of the groups produced by the first phase). In this way, the sample includes groups that represent both head (popular) and tail (rare) entities, in the same proportion as expected in the whole dataset.

Evaluation of the Steps Table 3 reports results on the contribution of each step of the RAF- STD approach. The Source Attribute Matching step clusters together source attributes that share the same values for a sample of linked specifications. If we launch this step once (Match, in Table 3), without doing any further processing, we get few false positives but a low recall. The approach is strongly conservative and produces very homogeneous clusters, achieving high precision but low recall.

In combination with the tagging step (Match+Tag, in Table 3), the overall approach improves the recall, thanks to the creation of virtual attributes that increment the match opportunities, with a small loss in precision, which is due to the possibility that some wrong virtual attributes produce accidental matches.

With the final step (Full algorithm, in Table 3), which merges clusters based on attribute names and domain similarity, the system significantly improves the recall, at the cost of a small loss in precision.

It is worth observing that the task addressed by the Source Attribute Matching step resembles that of a traditional schema matching solution, if we view the source attributes like the columns of relational tables. Compared to traditional schema matching approaches, our Source Attribute Matching algorithm has been designed to deal with a high degree of heterogeneity exploiting the availability of linkage information.

Table 4 Results of RAF- STD replacing the Source Attribute Matching algorithm with Coma++

	P	R	F1
Camera	0.58	0.70	0.63
Monitor	0.70	0.66	0.68

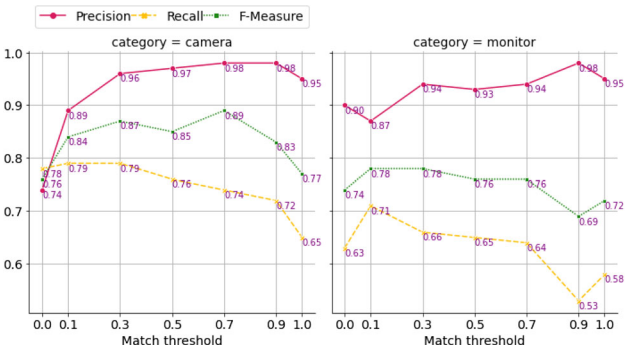

Fig. 5 Varying the match threshold

In order to evaluate the effectiveness of our approach, we have replaced in the RAF- STD system the module that implements our Source Attribute Matching algorithm with a traditional schema matching system. Namely, we used the instance-based version of Coma++, which according to the Valentine analysis [27] is the most performing approach.[18]

Table 4 reports the results of the experiment. The benefits of using the Source Attribute Matching instead of traditional schema matching algorithm are apparent, especially for the precision: by exploiting the linkage information, our algorithm can effectively contain false positive without penalizing the recall.

Robustness The source attribute matching algorithm considers that two source attributes match if the probability of match is above a threshold of 0.5, whose theoretical interpretation is that the conditional probability of equivalence $P(A \equiv B|L_{AB})$ is larger than that of non-equivalence $P(A \not\equiv B|L_{AB})$. It also assumes that the probability that a source provides a wrong value for a given triple (ϵ) is 0.1. We investigated the robustness of the approach varying these parameters.

Figure 5 shows the results of RAF- STD with different values of the matching threshold, i.e. the minimum score above which source attribute pairs are considered to match by the Bayesian analysis. The higher the threshold is, the more conservative the algorithm becomes: precision increases and recall decreases. The differences are, however, not so significant: in most cases the Bayesian analysis provides strong evidence of match or mismatch, making the algorithm robust with respect to this threshold. Also, precision never drops

[18] Namely, we used the COMA 3.0 Community Edition available on the Valentine repository (https://github.com/delftdata/valentine/tree/master/valentine/algorithms/coma).

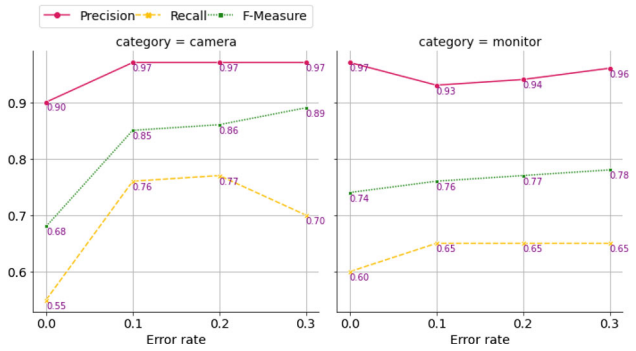

Fig. 6 Varying the error rate ϵ

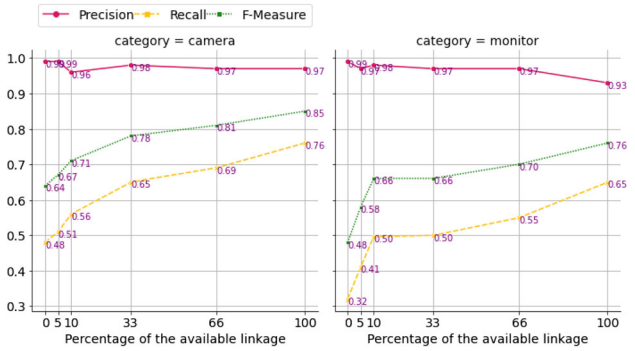

Fig. 7 Precision, recall and F-measure varying the amount of the linkage used as a percentage of total available linkage. Here, 100% represents the whole original sample, which is in turn about 14% of the estimated total linkage

even with a very low threshold: some low-weighted edges, even if above threshold, may be ignored if they break the assumption that no triples in the same specification can be in the same cluster.

Figure 6 shows the results of RAF- STD varying the error rate parameter ϵ between 0 and 0.3. In general, a high error rate favours pairs of source attributes with some mismatch (different values for specifications in linkage), while it penalizes pairs with no mismatch. However, the results of the overall approach generally remain quite stable, proving that in most cases the choice of a specific value for ϵ does not affect the results of the algorithm. Note that recall drops with $\epsilon = 0$: source attribute pairs having even just a single different value for a given product are considered as non-match, and precision does not necessarily improve: the algorithm incorrectly matches source attribute pairs with few distinct (and frequent) common values but on many instances.

The Role of Linkage The Bayesian analysis also exploits the linkage sample to match attributes, by comparing the values of attributes in linked instances. To evaluate the robustness of the algorithm with respect to the size of the linkage sample, we artificially removed a random part of the linkage and evaluated the performance of the approach.

Figure 7 reports the results of the experiment. We observe that without linkage, the recall worsen, thus confirming the ability of our approach to leverage the linkage information. It is interesting to observe that the recall drop is more pronounced on the Monitor dataset. Indeed, this vertical has many attributes, such as the number of dvi port or the number of USB port, with small and overlapping domains that can be easily confused without relying on linkage information. With a small amount (10%) of the available linkage (which is a sample whose size is estimated around 14% of the real linkage present in the datasets) the Bayesian matching step can rely on sufficient evidence to produce good recall, which continues to increase as the amount of linkage grows.

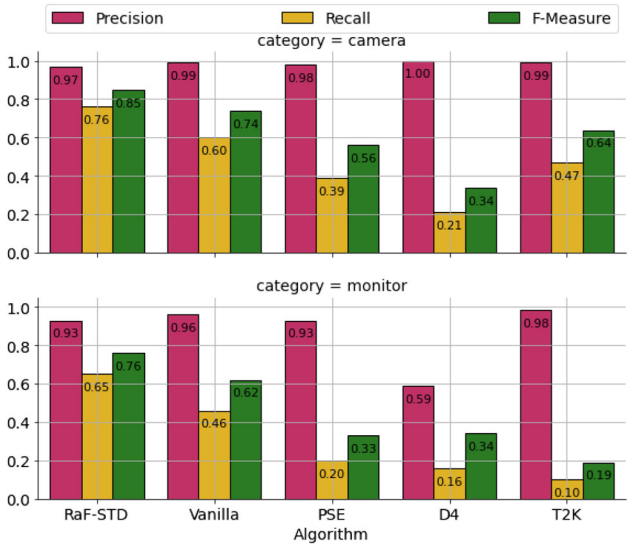

Fig. 8 Comparison with baselines and alternative approaches

6.2 Comparison with alternative approaches

We compared the RAF- STD approach to a *Vanilla* baseline and to three alternative approaches from the literature. The *Vanilla* baseline considers the names of the attributes and the similarity of their domains. Namely, it creates clusters of triples based on the similarity of the domain of the source attributes, using the Jaccard containment index as similarity measure; then, it merges the clusters that overlap with at least one attribute. The alternative approaches from the literature, as we described in Sect. 2, represent different solutions that can be applied to address the semantic type discovery problem: D4, T2K, and PSE.

Figure 8 shows the results of this comparison. All the approaches achieve high precision, demonstrating robustness over false positives. However, RAF- STD significantly outperforms all the competitors in recall, thus obtaining a better

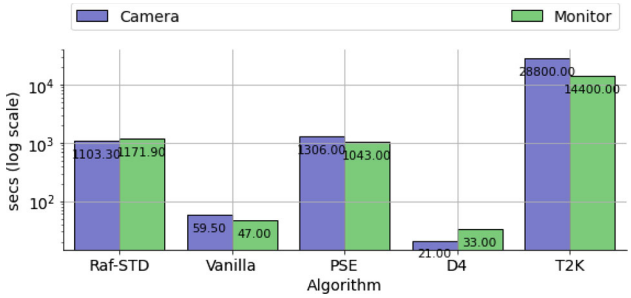

Fig. 9 Running times (seconds, in log scale)

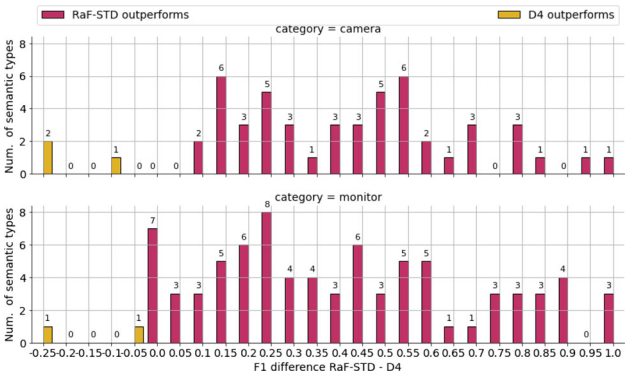

Fig. 10 Number of semantic types with a given F-measure difference between RAF- STD and D4, using an adaptation of D4 evaluation approach

F-measure on both the camera and the monitor datasets. [19] Figure 9 reports the running times of the different approaches. In Sect. 6.4, we illustrate in more details how RAF- STD scales with the number of sources.

Let us now comment in more detail the results obtained by the three competitive approaches.

Comparison to D4 D4 [34] is an unsupervised approach, making it a good candidate for comparison with RAF- STD. The algorithm discovers semantic types of columns or column subsets in a dataset.[20] However, its main goal is to discover semantic type domains (i.e. all possible values), and it is optimized for this goal. For example, it tends to exclude columns or column subsets if they do not provide new distinct values for the semantic type.

D4 provides an evaluation (precision, recall, F-Measure) specific to each semantic type. In particular, given a semantic type in the ground truth, for each cluster in their output that overlaps with at least one value, they compute precision, recall and F-Measure, and then keep the cluster with the best F-Measure. We compared with D4 using this metric too, but considering triples instead of semantic types values.

Figure 10 shows how many semantic types (vertical axis) have a given difference in F-measure between RAF- STD and D4 (horizontal axis). In many cases RAF- STD dominates, especially for numerical attributes (such as, `focal length max`) and descriptive types (such as, `auto focus mode` or `file system`). D4 works slightly better only for a few categorical and clean attributes, such as `color` and `external memory type`.

Comparison to PSE PSE [33] is a closed-world approach: target attributes (which can relate to semantic types) are limited to attributes present in a reference catalog of products provided as input. However, it does not need training data, and it can be easily adapted to discover semantic types.

To this end, we elected as *catalog* the source with most specifications in linkage and then aligned the other sources according to the PSE approach. In order to make it an open-world, data-driven approach, we do not delete source attributes that do not match with any attribute of the catalog (as in the original approach), but we add them to the catalog, so they are available for matching with further source attributes.

The main takeaway of this comparison is that the adaptation of this work to the semantic type discovery problem is not effective. Notice that PSE is unsupervised: its classifier is trained selecting attribute pairs with the same name as positive examples, relying on the assumption that attributes with the same name are semantically equivalent. In a heterogeneous setting, this choice compromises the accuracy of the predictions.

Comparison to T2K Also T2K [39] follows a closed-world approach, as it matches web table columns to target types present in a knowledge base (DBPedia). It does not need training data, so we adapted it similarly to what we did for PSE: we chose the source with most linkage as knowledge base, we processed the first source, then we created a new version of knowledge base merging these two sources, according to the T2K results for schema matching and record linkage. Then, in a similar way, we processed all other sources. Similarly to PSE, we have a good precision but recall is lower than with RAF- STD approach. Indeed, a closed-world approach that assumes clean and homogeneous knowledge base is not perfectly suited for our dataset.

6.3 Experiments on the WDC dataset

The WDC Product Data Corpus (WDC) is a publicly available dataset produced by extracting product offers from

[19] We also tested in isolation the two techniques that compose the *Vanilla* method: overlap of attribute names and Jaccard index across the domains. The F-measure obtained by the former on camera and monitor was 0.69 and 0.64, respectively. The Jaccard index produced 0.54 and 0.48, respectively. For the sake of readability, we do not report detailed results in the plots as they are not very significant.

[20] We configured it to take into account numerical values, which it does not by default.

CommonCrawl.org for 25 product categories. Each product offer provides the page title, a textual description and, in about 17% of cases, product specifications extracted from the page using HTML standard annotations like MicroData [35].

We have chosen three product categories, all with a rich set of specifications and significant linkage, and all dealing with completely different products from those in the DI2KG benchmark: Clothing, Jewelry and Automotive. It is worth noting that these are broad categories. For example, Jewelry includes products such as watches, necklaces, rings and many other types of personal ornaments; Clothing includes different types of dresses as well as coats, trousers and accessories. In WDC, many sources are very small, and many specifications contain only a few triples and are noisy because of extraction errors. To clean the dataset, we filtered out attributes that were not present in at least three specifications, specifications with less than three attributes, and sources with less than three specifications.

WDC does not provide ground truth for semantic type discovery, so we built it by ourselves. The linkage information furnished by the benchmark is partial and potentially noisy: it has been obtained grouping offers by product code (such as MPN or UPC) when it was present in the original pages. Unlike the DI2KG datasets, we do not have any information about the amount of available linkage as a proportion to the ground-truth linkage. In our experiments, we have used all the available linkage.

Figure 11 reports the results produced by RAF- STD and four alternative solutions on the WDC datasets. Let us comment each category.

Clothing The sources of this category often have attributes in different languages. RAF- STD correctly manages these attributes, comparing values for specifications in linkage. T2K also exploits linkage, but it is affected by attribute sparsity and does not exploit global redundancy of sources. The other approaches obtain a significantly lower recall. Indeed, they cannot rely on attribute names (because of differences in languages), nor on domains, because of the heterogeneity of the products and the sparsity of attributes.

Jewelry In Jewelry sources, many attributes have similar domains but different semantics; sometimes they are even related to different kinds of products (e.g. jewel material, bracelet material and, for watches, case material). RAF- STD correctly separates these attributes exploiting the linkage information, while the *Vanilla* baseline and D4, whose approaches are based on the similarity of the domains, tend to cluster these attributes together, thus obtaining lower precision. T2K exploits linkage information, keeping the precision very high, but the recall that it achieves is quite low. Because of the diversity of attribute names across sources (also due to the presence of different languages), the PSE approach does not have enough evidence for training the classifier. RAF- STD also correctly extracts brand name from attributes like product and packaging.

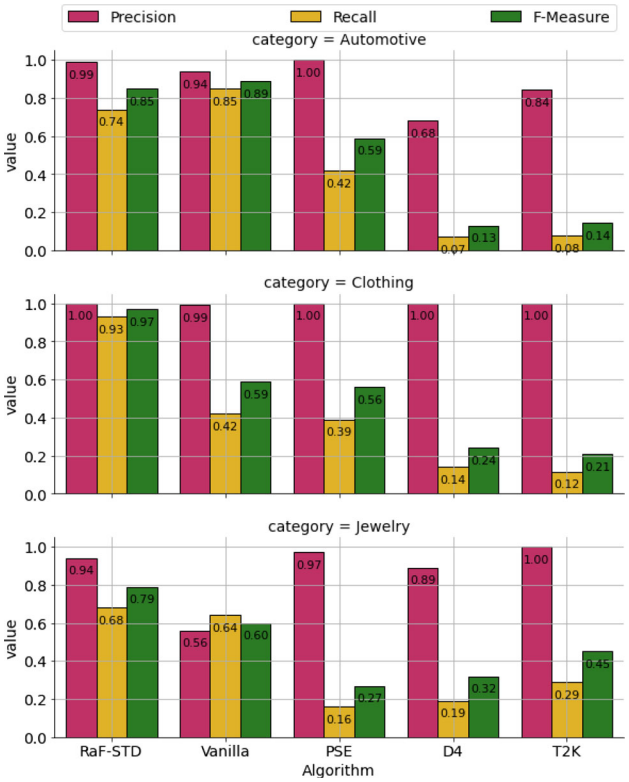

Fig. 11 Results on the WDC datasets

Automotive On the Automotive dataset, RAF- STD performs worse than the *Vanilla* baseline. Inspecting the results, we observed that the input linkage sample is limited and, most important, it does not really reflect the correspondences between the entities: it refers to the car model, while the offers describe individual (used) cars. Therefore many attributes (such as mileage and stock number) have completely different values, even between linked specifications. In Sect. 7, we discuss future work to overcome the issues that arise with ambiguous linkage. These linkage issues also affect T2K results, while the presence of a lot of numerical data with heterogeneous formats causes generally bad results for D4, which was not designed for numerical data.

Also for the WDC datasets we have investigated how the amount of linkage information impacts on the performances of the system. To this end, we have conducted an experiment running RAF- STD with 5%, 10% and 20% of the available linkage. The results are plotted in Fig. 12: similarly to the DI2KG datasets, the amount of link information provided as input to the system primarily affects recall, while precision is fairly stable. Overall, even with small amounts of linkage information the performances of the system are good.

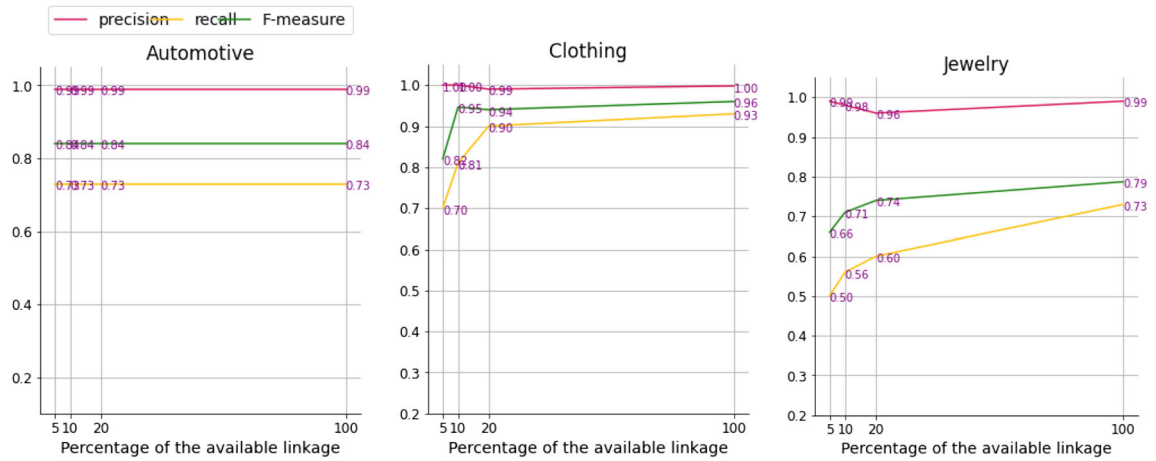

Fig. 12 Precision, recall and F-measure varying the amount of the linkage used as a percentage of total available linkage in the WDC dataset

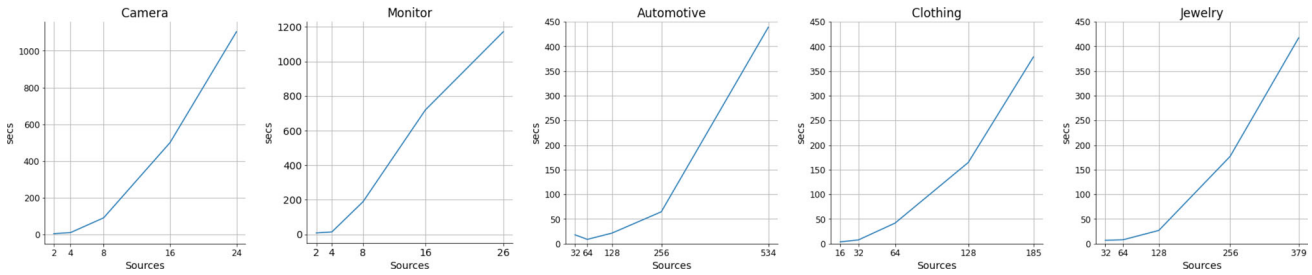

Fig. 13 Running time varying the number of sources on the WDC dataset

6.4 Scalability

In order to evaluate the scalability of the approach, we have run RAF- STD on a varying number of randomly picked sources. To avoid biases due to the random choice, fixed the number of sources, we have repeated the experiment 5 times, each one with a different set of randomly picked sources, and we have computed average F-measure and running time, removing best and worst cases.

Figure 13 presents the running times of RAF- STD as a function of the number of sources, for all the datasets of the DI2KG and WDC used in our evaluation. The plots show that the time is generally quadratic to the number of sources. Indeed, we proved this conjecture by computing the coefficient of determination (R^2) of the quadratic regression computed on each category, and we got very high results: more than 0.999 for each category, except for monitor which is 0.988. We can also infer that the system is quadratic to the number of source attributes, as the 5-times repetition avoids biases when we could choose very big or very small sources (in terms of number of source attribute). In Sect. 5.5, the complexity analysis concludes that our approach is cubic to the number of source attributes, assuming, in the worst case, the number of iterations being proportional to the number of source attributes. In practice, we observed that the number of iterations can be considered computationally constant: augmenting the number of sources, and consequently the number of source attributes, it quickly converges to a small number,[21] making the approach quadratic.

Figure 14 reports the quality of the results. We observe that RAF- STD exploits evidence that is brought by the data redundancy. In particular, observe that the F-measure improves with an increasing number of sources because of a better recall, while the impact on precision, due to noise and heterogeneity, is very low.

7 Conclusions and future work

We addressed the issue of discovering semantic types in multiple heterogeneous sources. We proposed a fine-grained approach, RAF- STD, in order to overcome the limitations of traditional approaches with heterogeneous and sparse sources.

[21] Profiling the execution of the system, we observed at most 3 iterations for all the datasets.

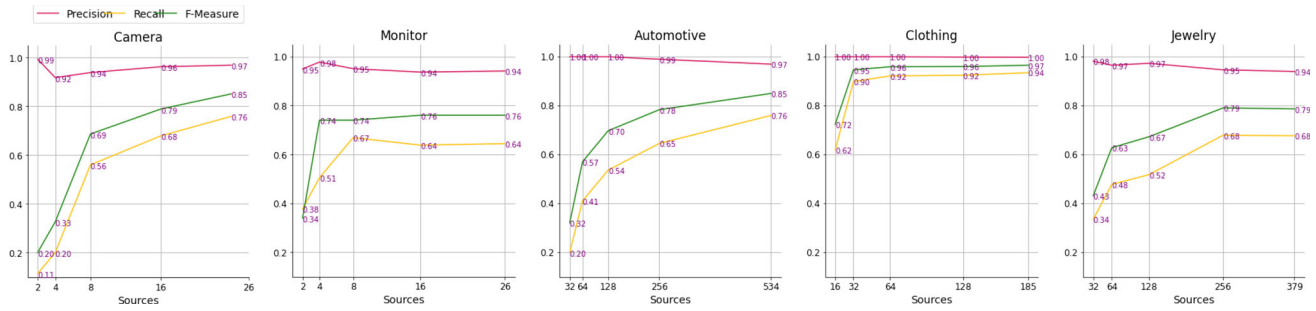

Fig. 14 Precision, recall, F-measure varying the number of sources on the WDC datasets

We performed extensive experiments using publicly available datasets, showing robustness and flexibility using different parameters and under different conditions. We analysed the contribution of each step of our approach to the final results and compared our results to other approaches, obtaining superior performance.

RAF- STD relies on record linkage information and expects that attributes for the same entity are consistent across sources, apart from some errors which are tolerated by the approach. However, the semantics of the linkage may happen to be in contrast with such an assumption. In the WDC Automotive category, we observed that the semantics of the input linkage information was related to the model of a car, while attributes such as mileage and stock number refer to the physical entity. Similar issues might occur considering the temporal evolution of an entity (e.g. consider the values of attributes for a company in 2015, and the values of the same attributes for the same company in 2020). A more detailed study on how to manage semantic type discovery with different record linkage semantics is an intriguing direction for future work.

The clusters of triples produced by RAF- STD are anonymous. Providing a name to each cluster may be a necessary step for downstream data processing tasks. Associating a meaningful name with each cluster can be done manually, but with a large number of clusters it might be quite expensive. To reduce the human effort of this step, an automatic procedure could suggest candidate names by considering the distribution of the attribute names of each cluster of triples. Investigating this opportunity is left to future work.

Funding Open access funding provided by Universitá degli Studi Roma Tre within the CRUI-CARE Agreement.

References

1. Abedjan, Z., Chu, X., Deng, D., Fernandez, R.C., Ilyas, I.F., Ouzzani, M., Papotti, P., Stonebraker, M., Tang, N.: Detecting data errors: Where are we and what needs to be done? PVLDB **9**(12), 993–1004 (2016)
2. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with coma++. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 906–908 (2005)
3. Banerjee, A., Krumpelman, C., Ghosh, J., Basu, S., Mooney, R.J.: Model-based overlapping clustering. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pp. 532–537 (2005)
4. Barbosa, L., Crescenzi, V., Dong, X.L., Merialdo, P., Piai, F., Qiu, D., Shen, Y., Srivastava, D.: Big data integration for product specifications. IEEE Data Eng. Bull. **41**(2), 71–81 (2018)
5. Bellahsene, Z., Bonifati, A., Rahm, E.: Schema Matching and Mapping. Springer Science & Business Media, Berlin (2011)
6. Berlin, J., Motro, A.: Autoplex: Automated discovery of content for virtual databases. In: International Conference on Cooperative Information Systems, pp. 108–122. Springer (2001)
7. Bhagavatula, C.S., Noraset, T., Downey, D.: Tabel: Entity linking in web tables. In: International Semantic Web Conference, pp. 425–441. Springer (2015)
8. Bilke, A., Naumann, F.: Schema matching using duplicates. In: 21st International Conference on Data Engineering (ICDE'05), pp. 69–80. IEEE (2005)
9. Bloom, S.A.: Similarity indices in community studies: potential pitfalls. Mar. Ecol. Prog. Ser **5**(2), 125–128 (1981)
10. Brunner, U., Stockinger, K.: Entity matching with transformer architectures-a step forward in data integration. In: International Conference on Extending Database Technology, Copenhagen, 30 March-2 April 2020 (2020)
11. Cannaviccio, M., Barbosa, D., Merialdo, P.: Towards annotating relational data on the web with language models. In: Proceedings of the 2018 World Wide Web Conference, pp. 1307–1316 (2018)
12. Chen, C., Golshan, B., Halevy, A.Y., Tan, W.C., Doan, A.: Biggorilla: An open-source ecosystem for data preparation and integration. IEEE Data Eng. Bull. **41**(2), 10–22 (2018)
13. Chu, X., Ilyas, I.F., Krishnan, S., Wang, J.: Data cleaning: Overview and emerging challenges. In: Proceedings of the 2016 ACM

SIGMOD International Conference on Management of Data, pp. 2201–2206 (2016)
14. Dalvi, N., Machanavajjhala, A., Pang, B.: An analysis of structured data on the web. PVLDB **5**(7), 680–691 (2012)
15. Doan, A., Domingos, P., Halevy, A.: Learning to match the schemas of data sources: A multistrategy approach. Machine Learn. **50**(3), 279–301 (2003)
16. Dong, X.L.: Challenges and innovations in building a product knowledge graph. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 2869–2869. ACM (2018)
17. Dong, X.L.: Building a broad knowledge graph for products. In: Proceedings of the 35th International Conference on Data Engineering (ICDE), pp. 25–25. IEEE (2019)
18. Dong, X.L., Srivastava, D.: Big data integration. Synthesis Lect. Data Manag. **7**(1), 1–198 (2015)
19. Engmann, D., Massmann, S.: Instance matching with coma++. In: BTW workshops, vol. 7, pp. 28–37 (2007)
20. Furche, T., Gottlob, G., Grasso, G., Guo, X., Orsi, G., Schallhart, C., Wang, C.: Diadem: thousands of websites to a single database. PVLDB **7**(14), 1845–1856 (2014)
21. Guo, C., Hedeler, C., Paton, N.W., Fernandes, A.A.: Matchbench: benchmarking schema matching algorithms for schematic correspondences. In: British National Conference on Databases, pp. 92–106. Springer (2013)
22. Hadjieleftheriou, M., Srivastava, D.: Approximate string processing. Foundations and Trends® in Databases **2**(4), 267–402 (2011)
23. Hulsebos, M., Hu, K., Bakker, M., Zgraggen, E., Satyanarayan, A., Kraska, T., Demiralp, Ç., Hidalgo, C.: Sherlock: A deep learning approach to semantic data type detection. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1500–1508 (2019)
24. Kang, J., Naughton, J.F.: On schema matching with opaque column names and data values. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 205–216 (2003)
25. Kannan, A., Givoni, I.E., Agrawal, R., Fuxman, A.: Matching unstructured product offers to structured product specifications. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 404–412. ACM (2011)
26. Kim, W., Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. Computer **24**(12), 12–18 (1991)
27. Koutras, C., Siachamis, G., Ionescu, A., Psarakis, K., Brons, J., Fragkoulis, M., Lofi, C., Bonifati, A., Katsifodimos, A.: Valentine: Evaluating matching techniques for dataset discovery. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 468–479. IEEE (2021)
28. Li, Y., Li, J., Suhara, Y., Doan, A., Tan, W.C.: Deep entity matching with pre-trained language models. PVLDB **14**(1), 50–60 (2020)
29. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. PVLDB **3**(1–2), 1338–1347 (2010)
30. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
31. Mausam, M.: Open information extraction systems and downstream applications. In: Proceedings of the twenty-fifth international joint conference on artificial intelligence, pp. 4074–4077 (2016)
32. Mork, P., Seligman, L., Rosenthal, A., Korb, J., Wolf, C.: The harmony integration workbench. In: Journal on Data Semantics XI, pp. 65–93. Springer (2008)
33. Nguyen, H., Fuxman, A., Paparizos, S., Freire, J., Agrawal, R.: Synthesizing products for online catalogs. PVLDB **4**(7), 409–418 (2011)
34. Ota, M., Müller, H., Freire, J., Srivastava, D.: Data-driven domain discovery for structured datasets. PVLDB **13**(7), 953–967 (2020)
35. Primpeli, A., Peeters, R., Bizer, C.: The wdc training dataset and gold standard for large-scale product matching. In: Companion Proceedings of The 2019 World Wide Web Conference, pp. 381–386 (2019)
36. Qiu, D., Barbosa, L., Crescenzi, V., Merialdo, P., Srivastava, D.: Big data linkage for product specification pages. In: Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data, pp. 67–81. ACM (2018)
37. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4), 334–350 (2001)
38. Ritze, D., Bizer, C.: Matching web tables to dbpedia-a feature utility study. Context **42**(41), 19–31 (2017)
39. Ritze, D., Lehmberg, O., Bizer, C.: Matching html tables to dbpedia. In: Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, pp. 1–6 (2015)
40. Schubert, A.: Measuring the similarity between the reference and citation distributions of journals. Scientometrics **96**(1), 305–313 (2013)
41. Sekhavat, Y.A., Di Paolo, F., Barbosa, D., Merialdo, P.: Knowledge base augmentation using tabular data. In: LDOW (2014)
42. Tang, N., Fan, J., Li, F., Tu, J., Du, X., Li, G., Madden, S., Ouzzani, M.: Rpt: relational pre-trained transformer is almost all you need towards democratizing data preparation. Proc. VLDB Endowment **14**(8), 1254–1261 (2021)
43. Yan, C., He, Y.: Synthesizing type-detection logic for rich semantic data types using open-source code. In: Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data, pp. 35–50 (2018)
44. Zhang, D., Li, D., Guo, L., Tan, K.L.: Unsupervised entity resolution with blocking and graph algorithms. IEEE Trans. Knowledge Data Eng. (2020)
45. Zhang, D., Suhara, Y., Li, J., Hulsebos, M., Demiralp, C., Tan, W.C.: Sato: Contextual semantic type detection in tables. PVLDB **13**(11) (2019)
46. Zhang, S., Balog, K.: Web table extraction, retrieval, and augmentation: a survey. ACM Trans. Intell. Syst. Technol. (TIST) **11**(2), 1–35 (2020)