



**HAL**  
open science

# An Overview of End-to-End Entity Resolution for Big Data

Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, Kostas Stefanidis

► **To cite this version:**

Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, Kostas Stefanidis. An Overview of End-to-End Entity Resolution for Big Data. ACM Computing Surveys, In press. hal-02955445

**HAL Id: hal-02955445**

**<https://hal.science/hal-02955445>**

Submitted on 6 Oct 2020

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# An Overview of End-to-End Entity Resolution for Big Data 1

VASSILIS CHRISTOPHIDES, ENSEA, ETIS Lab, France  
 VASILIS EFTHYMIU, IBM Research, USA  
 THEMIS PALPANAS, Université de Paris and French University Institute (IUF), France  
 GEORGE PAPADAKIS, National and Kapodistrian University of Athens, Greece  
 KOSTAS STEFANIDIS, Tampere University, Finland

One of the most critical tasks for improving data quality and increasing the reliability of data analytics is *Entity Resolution* (ER), which aims to identify different descriptions that refer to the same real-world entity. Despite several decades of research, ER remains a challenging problem. In this survey, we highlight the novel aspects of resolving Big Data entities when we should satisfy more than one of the Big Data characteristics simultaneously (i.e., Volume and Velocity with Variety). We present the basic concepts, processing steps, and execution strategies that have been proposed by database, semantic Web, and machine learning communities in order to cope with the loose *structuredness*, extreme *diversity*, high *speed*, and large *scale* of entity descriptions used by real-world applications. We provide an end-to-end view of ER workflows for Big Data, critically review the pros and cons of existing methods, and conclude with the main open research directions.

Q1

CCS Concepts 12

Additional Key Words and Phrases: Entity blocking and matching, strongly and nearly similar entities, block processing, batch and incremental entity resolution workflows, crowdsourcing, deep learning 13-14

**ACM Reference format:** 15

Vassilis Christophides, Vasilis Eftymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 53, 6, Article 127 (September 2020), 42 pages. <https://doi.org/10.1145/3418896> 16-19

## 1 INTRODUCTION 21

In the Big Data era, business, government, and scientific organizations increasingly rely on massive amounts of data collected from both internal (e.g., CRM, ERP) and external data sources (e.g., the Web). Even when data integrated from multiple sources refer to the same real-world entities, they usually exhibit several quality issues such as *incompleteness* (i.e., partial data), *redundancy* (i.e., overlapping data), *inconsistency* (i.e., conflicting data), or simply *incorrectness* (i.e., data errors). A typical task for improving various aspects of data quality is *Entity Resolution* (ER). 22-27

This work was partially funded by the EU H2020 project ExtremeEarth (825258).  
 Authors' addresses: V. Christophides, ENSEA, ETIS Lab, France; email: vassilis.christophides@inria.fr; V. Eftymiou, IBM Research; email: vasilis.efthymiou@ibm.com; T. Palpanas, Université de Paris and French University Institute (IUF), France; email: themis@mi.parisdescartes.fr; G. Papadakis, National and Kapodistrian University of Athens, Greece; email: gpapadis@di.uoa.gr; K. Stefanidis, Tampere University, Finland; email: konstantinos.stefanidis@tuni.fi  
 Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).  
 © 2020 Association for Computing Machinery.  
 0360-0300/2020/09-ART127 \$15.00  
<https://doi.org/10.1145/3418896>

Q2



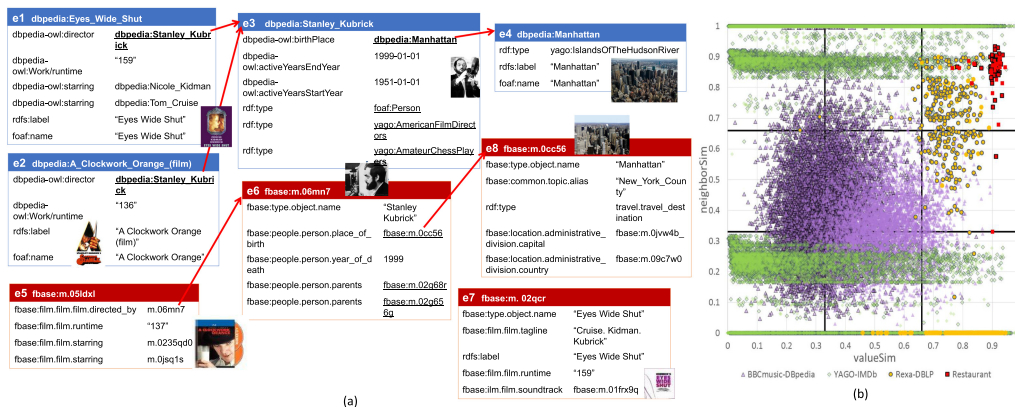


Fig. 1. (a) Movies, directors, and locations from DBpedia (blue) and Freebase (red), where  $e_1, e_2, e_3,$  and  $e_4$  match with  $e_7, e_5, e_6,$  and  $e_8,$  respectively. (b) Value and neighbor similarity distribution of matches in four datasets.

ER aims to identify different descriptions that refer to the same real-world entity appearing either within or across data sources, when unique entity identifiers are not available. Typically, ER aims to match structured descriptions (i.e., records) stored in the same (a.k.a., *deduplication*), or two different (a.k.a., *record linkage*) relational tables. In the Big Data era, other scenarios are also considered, such as matching semi-structured descriptions across RDF knowledge bases (KBs) or XML files (a.k.a., *link discovery* or *reference reconciliation*). Figure 1(a) illustrates descriptions of the same movies, directors, and places from two popular KBs: DBpedia (blue) and Freebase (red). Each entity description is depicted in a tabular format, where the header row is the URI of the description and the remaining rows are the attribute (left) -value (right) pairs of the description.

ER aims to classify pairs of descriptions that are assumed to correspond to the same (vs. different) entity into *matches* (vs. *non-matches*). An ER process usually encompasses several tasks, including *Indexing* (a.k.a., *Blocking*), which reduces the number of candidate descriptions to be compared in detail, and *Matching*, which assesses the similarity of pairs of candidate descriptions using a set of functions. Several ER frameworks and algorithms for these tasks have been proposed during the last three decades in different research communities. In this survey, we present the latest developments in ER, explaining how the Big Data characteristics call for novel ER frameworks that relax a number of assumptions underlying several methods and techniques proposed in the context of the database [34, 50, 58, 106, 124], machine learning [72] and semantic Web communities [127].

Our work is inspired by the Linked Open Data (LOD) initiative [37], which covers only a small fragment of the Web today, but is representative of the challenges raised by Big Data to core ER tasks: (a) how descriptions can be effectively compared for similarity, and (b) how resolution algorithms can efficiently filter the number of candidate description pairs that need to be compared.

**Big Data Characteristics.** Entity descriptions published as LOD exhibit the 4 “V”s [49] that challenge existing individual ER algorithms, but also entire ER workflows:

- *Volume.* The content of each data source never ceases to increase and so does the *number of data sources*, even for a single domain. For example, the LOD cloud currently contains more than 1,400 datasets from various sources (this is a  $\times 100$  growth since its first edition) in 10 domains with  $>200B$  triples (i.e.,  $\langle \text{subject}, \text{predicate}, \text{object} \rangle$ ) describing more than 60M entities of different types<sup>1</sup>; the life-science domain alone accounts for  $>350$  datasets.

<sup>1</sup><https://lod-cloud.net>.

- *Variety*. Data sources are extremely heterogeneous, even in the same domain, regarding both how they structure their data and how they describe the same real-world entity. In fact, they exhibit *considerable diversity* even for substantially similar entities. For example, there are ~700 vocabularies in the LOD cloud, but only ~100 of them are shared by more than 1KB.<sup>2</sup>
- *Velocity*. As a direct consequence of the rate at which data is being collected and continuously made available, many of the data sources are *very dynamic*. For example, LOD data are rarely static, with recent studies reporting that 23% of the datasets exhibit infrequent changes, while 8% are highly dynamic in terms of triples additions and deletions.<sup>3</sup>
- *Veracity*. Data sources are of *widely differing quality*, with significant differences in the coverage, accuracy, and timeliness of data provided. Even in the same domain, various forms of inconsistencies and errors in entity descriptions may arise, due to the limitations of the automatic extraction techniques, or of the crowd-sourced contributions. A recent empirical study [44] shows that there are several LOD quality problems, as their conformance with a number of best practices and guidelines is still open. For example, in Figure 1(a), the descriptions of “A Clockwork Orange” from DBpedia ( $e_2$ ) and Freebase ( $e_3$ ) differ in their runtime.

**Big Data Entity Resolution.** Individual characteristics of Big Data have been the focus of previous research work in ER. For example, there is a continuous concern for improving the *scalability* of ER techniques over increasing *Volumes* of entities using massively parallel implementations [29]. Moreover, uncertain entity descriptions due to high *Veracity* have been resolved using approximate matching [50, 69]. However, traditional deduplication techniques [35, 58] have been mostly conceived for processing structured data of few entity types after being adequately pre-processed in a data warehouse, and hence been able to discover blocking keys of entities and/or mapping rules between their types. We argue that ER techniques are challenged when more than one of the Big Data “V”s have to be addressed simultaneously (e.g., *Volume* or *Velocity* with *Variety*).

In essence, the high *Variety* of Big Data entities calls for a paradigm shift in all major tasks of ER. Regarding *Blocking*, *Variety* renders inapplicable the traditional techniques that rely on schema and domain knowledge to maximize the number of comparisons that can be skipped, because they do not lead to matches [133]. As far as *Matching* is concerned, *Variety* requires novel entity matching approaches that go beyond approximate string similarity functions [107]. This is because such functions are applied on the values of specific attributes among pairs of descriptions, which are difficult to be known in advance. Clearly, *schema-aware* comparisons cannot be used for *loosely structured and highly heterogeneous entity descriptions*, such as those found in LOD. Similarity evidence of entities can be obtained only by looking at the bag of literals contained in descriptions; regardless of the attributes, they appear as values. Finally, as the *value-based* similarity of a pair of entities may still be weak due to *Veracity*, we need to consider additional sources of matching evidence related to the *similarity of neighboring* entities, which are connected via relations.

The previous challenges are exemplified in Figure 1(b), which depicts the two types of similarity for entities known to match from four established benchmark datasets: Restaurant,<sup>4</sup> Rexa-DBLP,<sup>5</sup> BBCmusic-DBpedia,<sup>6</sup> and YAGO-IMDb.<sup>7</sup> Every dot corresponds to a different matching pair, while its shape denotes the respective dataset. The horizontal axis reports the normalized value similarity

<sup>2</sup><https://lov.linkeddata.es/dataset/lov>.

<sup>3</sup><http://km.aifb.kit.edu/projects/dyldo>.

<sup>4</sup><http://oaei.ontologymatching.org/2010/im>.

<sup>5</sup><http://oaei.ontologymatching.org/2009/instances>.

<sup>6</sup><http://datahub.io/dataset/bbc-music>, <http://km.aifb.kit.edu/projects/btc-2012>.

<sup>7</sup><http://www.yago-knowledge.org>, <http://www.imdb.com>.

98 based on the common words in a pair of descriptions (weighted Jaccard [111]), while the vertical  
99 one reports the maximum value similarity of their respective entity neighbors. We can observe that  
100 the value-based similarity of matching entities significantly varies across different datasets. For  
101 *strongly similar entities* (e.g., value similarity  $>0.5$ ), existing duplicate detection techniques work  
102 well, but to resolve *nearly similar entities* (e.g., value similarity  $<0.5$ ), we need advanced ways of  
103 exploiting evidence about the similarity of neighboring entities, due to the Variety in entity types.

104 Additional challenges are raised by the *Velocity* of Big Data Entities. Even though ER is historically  
105 framed as an offline task that improves data quality in data warehouses upon completion of  
106 data integration, many services now require one to *resolve entities in real time*. Such services strive  
107 for incremental ER workflows over *dynamic sources* that can sacrifice completeness of the resulting  
108 matches as long as *query-based* [5, 17] or *streaming* [96] execution strategies can be supported.

109 **Contributions.** Record linkage and deduplication techniques for structured data in data ware-  
110 house settings are the subject of numerous surveys and benchmarking efforts [34, 35, 54, 58, 80, 87,  
111 106, 124]. Approximate instance matching is surveyed in [50], link discovering algorithms in [127],  
112 and uncertain ER in [69]. Recent efforts to enhance scalability of ER methods by leveraging dis-  
113 tribution and parallelization techniques are surveyed in [29], while overviews of blocking and  
114 filtering techniques are presented in [132, 140]. In contrast, our goal is to present an in-depth sur-  
115 vey on all tasks required to implement complex ER workflows, including Indexing, Matching, and  
116 Clustering.

117 To the best of our knowledge, this is the first survey that provides an end-to-end view of ER  
118 workflows for Big Data entities and of the new entity methods addressing the *Variety* in conjunc-  
119 tion with the *Volume* or the *Velocity* of Big Data Entities. Throughout this survey, we present  
120 the basic concepts, processing tasks, and execution strategies required to cope with the loose  
121 *structuredness*, extreme structural *diversity*, high *speed*, and large *scale* of entity descriptions ac-  
122 tually consumed by Big Data applications. This survey is intended to provide a starting point for  
123 researchers, students, and developers interested in recent advances of schema-agnostic, budget-  
124 aware, and incremental ER techniques that resolve nearly similar entity descriptions published by  
125 numerous Big Data sources.

126 The remaining of this survey is organized as follows. Section 2 presents the core concepts and  
127 tasks for building end-to-end ER workflows. Each workflow task is then examined in a separate  
128 section: Blocking in Section 3, Block Processing in Section 4, Matching in Section 5, and Clustering  
129 in Section 6. All these sections study methods for batch ER, while budget-aware and incremental  
130 ER are described in Sections 7 and 8, respectively. Section 9 covers complementary ER methods  
131 along with the main systems for end-to-end ER, while Section 10 elaborates on the most important  
132 directions for future work. Finally, Section 11 summarizes the current status of ER research.

133 Note that two of the authors have also published a survey on blocking and filtering (similarity  
134 join) techniques for structured and semi-structured data [140], which covers only two steps of  
135 the end-to-end ER workflow for Big Data entities: Blocking in Section 3 and Block Processing  
136 in Section 4. In contrast, this survey covers the entire end-to-end ER workflow, including Entity  
137 Matching, Clustering, and topics such as budget-aware, incremental, crowd-sourced, rule-based,  
138 deep learning-based, and temporal ER. The overlap of the two surveys is kept to the minimum.

## 139 2 ER PROCESSING TASKS AND WORKFLOWS

140 The core notion of *entity description* comprises a set of attribute-value pairs uniquely identified  
141 through a global id. A set of such descriptions is called *entity collection*. Two descriptions that are  
142 found to correspond to the same real-world object are called *matches* or *duplicates*. Depending on  
143 the input and its characteristics, the ER problem is distinguished into [56, 136, 153, 161]:

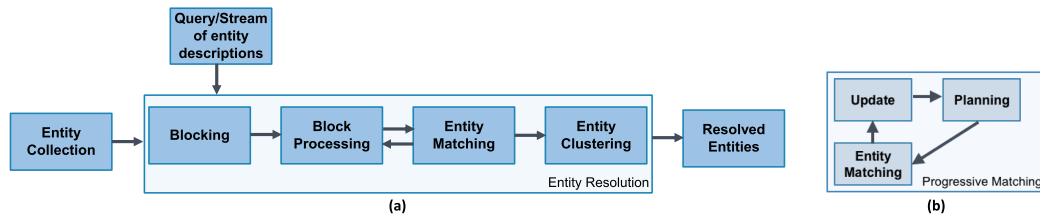


Fig. 2. (a) The generic end-to-end workflow for Entity Resolution. (b) Budget-aware Matching.

- (1) *Clean-Clean ER*, when the input comprises two overlapping, but individually clean (i.e., duplicate-free) entity collections and the goal is to find the matches between them. 144
- (2) *Dirty ER*, where the goal is to identify the duplicates within a single entity collection. 145
- (3) *Multi-source ER*, when more than two entity collections are given as input. 146

All previous instances of the ER problem involve general processing tasks as illustrated in the end-to-end workflow of Figure 2(a) [37, 166]. As every description should be compared to all others, the ER problem is by nature quadratic to the size of the input entity collection(s). To cope with large Volumes of entities, *Blocking* (a.k.a., *Indexing*) is typically applied as a first processing task to discard as many comparisons as possible without missing any matches. It places similar descriptions into blocks, based on some criteria (typically, called *blocking keys*) so that it suffices to execute comparisons only between descriptions co-occurring in at least one block. In other words, *Blocking* discards comparisons between descriptions that are unlikely to match, quickly splitting the input entity collection into blocks as close as possible to the final ER result. 148

To address *Variety* in Big Data, *Blocking* operates in a schema-agnostic fashion that considers all attribute values, regardless of the associated attribute names [141]. The key is *redundancy*, i.e., the act of placing every entity into multiple blocks, thus increasing the likelihood that matching entities co-occur in at least one block. On the flip side, the number of executed comparisons is extremely big. This is addressed, though, by a second processing task, called *Block Processing*. Its goal is to restructure an existing block collection so as to minimize the number of comparisons, without any significant impact on the duplicates that co-occur in blocks. This is achieved by discarding two types of unnecessary comparisons: the *redundant* ones, which are repeated across multiple blocks and the *superfluous* ones, which involve non-matching entities. 150

The next task is *Matching*, which, in its simplest form, applies a function  $M$  that maps each pair of entity descriptions  $(e_i, e_j)$  to  $\{true, false\}$ , with  $M(e_i, e_j) = true$  meaning that  $e_i$  and  $e_j$  are matches, and  $M(e_i, e_j) = false$  that they are not. Typically, the match function is defined via a similarity function  $sim$  that measures how similar two descriptions are to each other, according to certain comparison criteria. Finding a similarity function that perfectly distinguishes all matches from non-matches for all entity collections is rather hard. Thus, in reality, we seek a similarity function that is only good enough, minimizing the number of false-positive or -negative matches. 151

Recent works have also proposed an *iterative ER process*, which interleaves *Matching* with *Blocking* [148, 194]: *Matching* is applied to the results of (Meta-)Blocking and the results of each iteration potentially alter the existing blocks, triggering a new iteration. The block modifications are based on the relationships between the matched descriptions and/or on the results of their merging. 152

The final task in the end-to-end ER workflow is *Clustering* [80, 126, 153–155], which groups together the identified matches such that all descriptions within a cluster match. Its goal is actually to infer indirect matching relations among the detected pairs of matching descriptions so as to overcome possible limitations of the employed similarity functions. Its output comprises disjoint sets of entity descriptions  $R = \{r_1, r_2, \dots, r_m\}$ , such that (i)  $\forall e_i, e_j \in r_k M(e_i, e_j) = true$ , (ii)  $\forall e_i \in$  153

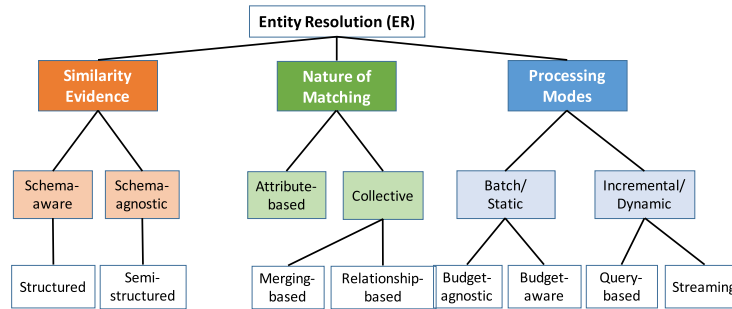


Fig. 3. Taxonomy of ER settings and approaches.

182  $r_k \forall e_j \in r_l M(e_i, e_j) = false$ , and (iii)  $\cup_{r_i} r_i \in R = \mathcal{E}$ , where  $\mathcal{E}$  stands for the input entity collection.  
 183 This partitioning corresponds to the resulting set of resolved entities in Figure 2(a).

184 Figure 2(b) illustrates the additional processing tasks that are required when an ER workflow  
 185 is subject to budget restrictions in terms of time or number of comparisons. These restrictions essentially  
 186 call for an approximate solution to ER, as an indirect way of addressing Volume. Rather  
 187 than finding all entity matches, the goal of *budget-aware ER* is to progressively identify as many  
 188 matches as possible within a specified cost budget. It extends batch, budget-agnostic ER workflows  
 189 with a *Planning* and *Update* phase that typically work on windows [2]. Planning is responsible for  
 190 selecting *which* pairs of descriptions will be compared for matching and in *what order*, based on  
 191 the cost/benefit tradeoff. Within every window, it essentially favors the more promising compar-  
 192 isons, which are more likely to increase the targeted benefit (e.g., the number of matches) in  
 193 the remaining budget. Those comparisons are performed first in the current window and thus, a  
 194 higher benefit is achieved as early as possible. The Update phase takes into account the results  
 195 of Matching, such that Planning in a subsequent window will promote the comparison of pairs  
 196 influenced by the previous matches. This iterative ER process continues until the budget is ex-  
 197 hausted. Both phases rely on a graph of dependencies among descriptions [48], which leverages  
 198 budget-agnostic blocking methods.

199 Finally, to resolve in real time entities provided as queries against a known entity collection,  
 200 or arriving in high Velocity streams, *incremental ER* workflows should be supported. In the first  
 201 case, a *summarization* of the entity collection can reduce the number of comparisons between a  
 202 query description and an indexed entity collection, by keeping—ideally in memory—representative  
 203 entity descriptions for each set of already resolved descriptions [96]. Thus, each query (description)  
 204 corresponds either to descriptions already resolved to a distinct real-world entity, or to a new one, if  
 205 it does not match with any other description [17, 164, 191]. To boost time efficiency, ER workflows  
 206 should support *dynamic indexing/blocking* at varying latencies and thus be able to compare only  
 207 a small number of highly similar candidate pairs arriving in a streaming fashion. Fast algorithms  
 208 are also required to incrementally cluster the graph formed by the matched entities in a way that  
 209 approximates the optimal performance of correlation clustering [77].

210 **Taxonomy of ER settings and approaches.** Overall, Figure 3 illustrates the taxonomy of ER set-  
 211 tings based on the key characteristics. Blocking, Matching, and Clustering methods that operate  
 212 on relational data are *schema-aware*, as opposed to the *schema-agnostic* methods, which are more  
 213 flexible regarding the structure, since they consider all attribute values. In the context of Big Data,  
 214 nearly similar entities are resolved by going beyond *attribute-based* ER techniques, which exam-  
 215 ine each pair of descriptions independently from other pairs. To match graph-based descriptions  
 216 of real-world entities, *collective* ER techniques [16] are used to increase their matching evidence

either by merging partially matched descriptions of entities or by propagating their similarity to neighbor entities via relations that will be matched in a next round. These techniques involve several iterations until they converge to a stable ER result (i.e., no more matches are identified). Thus, collective ER is hard to scale, especially in a cross-domain setting that entails a very large number of sources and entity types. Finally, we distinguish between *batch* (or *static*) ER, which operates on a given input entity collection, and *incremental* (or *dynamic*) ER, which operates on entities arriving in streams or provided by users online as queries. A fine-grained classification of the previous ER settings and approaches will be presented in the following subsections.

### 3 BLOCKING

This step receives as input one or more entity collections and returns as output a set of blocks  $\mathcal{B}$ , called *block collection*, which groups together similar descriptions, while keeping apart the dissimilar ones. As a result, each description can be compared only to others placed within the same block(s), thus reducing the computational cost of ER to the comparison of similar descriptions. Blocking is thus crucial for successfully addressing the Volume of Big Data.

The desiderata of Blocking are [35] (i) to place all matching descriptions in at least one common block, and (ii) to minimize the number of suggested comparisons. The second goal dictates skipping many comparisons, possibly leading to many missed matches, which hampers the first goal. Therefore, Blocking should achieve a good tradeoff between these two competing goals.

In this survey, we provide an overview of Blocking for semi-structured data, which require no domain or schema knowledge, unlike the schema-aware methods that are crafted for structured data (we refer the interested reader to [34, 35, 140] for more details). Instead of relying on human intervention, they require no expertise to identify the best attribute(s) for defining blocking keys. They operate in a *schema-agnostic* way that disregards the semantic equivalence of attributes, thus being inherently crafted for addressing the Variety of highly heterogeneous semi-structured data. We distinguish them into non-learning and learning-based methods.

**Non-learning methods.** *Semantic Graph Blocking* [131] considers exclusively the relations between descriptions, i.e., foreign keys in databases and links in RDF data. For every description  $e_i$ , it creates a block  $b_i$  that contains all descriptions connected with  $e_i$  through a path of restricted length, provided that the block size does not exceed a predetermined limit.

The textual content of attributes is considered by *Token Blocking (TB)* [136], which creates a block  $b_t$  for every distinct attribute value token  $t$ , regardless of the associated attribute names: two descriptions co-occur in  $b_t \in \mathcal{B}$ , if they share token  $t$  in any of their attribute values. This crude operation yields high recall, due to *redundancy* (i.e., every entity participates in multiple blocks), at the cost of low precision. This is due to the large portion of *redundant comparisons*, which are repeated in different blocks, and *superfluous* ones, which involve non-matching entities [133, 136, 138].

Discarding these two types of comparisons, especially the superfluous ones, we can raise TB's precision without any (significant) impact on recall. *Attribute Clustering Blocking* [136] clusters together attributes with similar values and applies TB independently to the values of every attribute cluster. *RDFKeyLearner* [165] applies TB independently to the values of automatically selected attributes, which combine high value discriminability with high description coverage. *TYPiMatch* [116] clusters the input descriptions into a set of overlapping types and then applies TB independently to the members of each type. Unlike TB, which tokenizes URIs on all their special characters, *Prefix-Infix(-Suffix) Blocking* [135] uses as blocking keys only the infixes of URIs—the *prefix* describes the domain of the URI, the *infix* is a local identifier, and the optional *suffix* contains details about the format, or a named anchor. For example, in “<https://dl.acm.org/journal/csur/authors>,” the prefix is “<https://dl.acm.org/journal/>,” the infix is “*csur*,” and the suffix is “*authors*.”



264 Another family of Blocking methods stems from generalizing TB’s functionality to the main  
 265 schema-aware non-learning techniques. By using the same blocking keys as TB, we can apply  
 266 traditional Blocking methods to heterogeneous semi-structured data [133] and significantly im-  
 267 prove their recall, even over structured data. This has been successfully applied to the following  
 268 techniques.

269 *Suffix Arrays Blocking* [1] converts each TB blocking key (i.e., attribute value token) into the  
 270 suffixes that are longer than a specific minimum length  $l_{min}$ . Then, it defines a block for every suf-  
 271 fix that does not exceed a predetermined frequency threshold  $b_{max}$ , which specifies the maximum  
 272 block size. *Extended Suffix Arrays Blocking* [35, 133] considers all substrings (not just the suffixes)  
 273 of TB blocking keys with more than  $l_{min}$  characters, so as to support noise at the end of blocking  
 274 keys (e.g., “JohnSnith” and “JohnSmith”). Similarly, *Q-grams Blocking* [35, 133] converts every TB  
 275 blocking key into sub-sequences of  $q$  characters (*q-grams*) and defines a block for every distinct  
 276 *q-gram*. *Extended Q-Grams Blocking* [35, 133] concatenates multiple *q-grams* to form more distinc-  
 277 tive blocking keys.

278 *Canopy Clustering* [35, 118] iteratively selects a random description  $e_i$  and creates a new block  
 279  $b_i$  for it. Using a cheap string similarity measure, it places in  $b_i$  all descriptions whose TB blocking  
 280 keys have a similarity to  $e_i$  higher than  $t_{in}$ ; descriptions with a similarity higher than  $t_{ex} (> t_{in})$  par-  
 281 ticipate in no subsequent block. *Extended Canopy Clustering* [35, 133] replaces the weight thresh-  
 282 olds with cardinality ones: for each randomly selected description, the  $k_1$  most similar descriptions  
 283 are placed in its block, while the  $k_2 (\leq k_1)$  most similar ones participate in no other block.

284 Finally, *Sorted Neighborhood* [84] sorts TB blocking keys in alphabetical order. A window of fixed  
 285 size  $w$  slides over the sorted list of descriptions and compares the description at the last position  
 286 with all descriptions in the same window. This approach is robust to noise in blocking keys, but  
 287 small  $w$  trades high precision for low recall and vice versa for large  $w$  [35]. To address this issue,  
 288 *Extended Sorted Neighborhood* [35, 133] slides the window  $w$  over the sorted list of *blocking keys*.

289 **Learning-based methods.** *Hetero* [100] is an unsupervised approach that maps every dataset to  
 290 a normalized TF vector, and applies an efficient adaptation of the Hungarian algorithm to pro-  
 291 duce positive and negative feature vectors. Then, it applies *FisherDisjunctive* [99] with bagging  
 292 to achieve robust performance. *Extended DNF BSL* [101] combines an established instance-based  
 293 schema matcher with weighted set covering to learn supervised blocking schemes in Disjunctive  
 294 Normal Form (DNF) with at most  $k$  attributes.

295 **Parallelization.** Parallel adaptations of the above methods have been proposed in the literature.  
 296 They rely on the *MapReduce paradigm* [43]: following a split-apply-combine strategy, MapReduce  
 297 partitions the input data into smaller chunks, which are then processed in parallel. A Map function  
 298 emits intermediate (key, value) pairs for each input split, while a Reduce function processes the list  
 299 of values that correspond to a particular intermediate key, regardless of the mapper that emitted  
 300 them. The two functions form a MapReduce job, with complex procedures involving multiple jobs.

301 Using a single MapReduce job, TB builds an inverted index that associates every token with all  
 302 entities containing it in their attribute values [37, 57]. For Attribute Clustering, four MapReduce  
 303 jobs are required [37, 57]: the first one aggregates all values per attribute, the second one estimates  
 304 the similarity between all attributes, the third one associates every attribute with its most similar  
 305 one, and the fourth one assigns to every attribute a cluster id and applies the TB MapReduce job.  
 306 Prefix-Infix(-Suffix) Blocking requires three jobs [37, 57]: the first two extract the prefixes and the  
 307 optional suffixes from the input URIs, respectively, while the third one applies TB’s mapper to the  
 308 literal values and a specialized mapper that extracts infixes to the URIs.

309 A crucial aspect of the MapReduce paradigm is the *load balancing algorithm*. To balance the cost  
 310 of executing the comparisons defined in an existing block collection, *Dis-Dedup* [38] formalizes

Table 1. A Taxonomy of the Blocking Methods Discussed in Section 3 (in the Order of Presentation)

	Indexing Function Definition		Redundancy attitude	
	non-learning	learning-based	redundancy-positive	redundancy-neutral
Semantic Graph Blocking [131]	✓			✓
Token Blocking [136]	✓		✓	
Attribute Clustering Blocking [136]	✓		✓	
Prefix-Infix(-Suffix) Blocking [135]	✓		✓	
Suffix Arrays Blocking [1]	✓		✓	
Extended Suffix Arrays Blocking [35,133]	✓		✓	
Q-Grams Blocking [35,133]	✓		✓	
Extended Q-Grams Blocking [35,133]	✓		✓	
Canopy Clustering [35,118]	✓			✓
Extended Canopy Clustering [35,133]	✓			✓
Sorted Neighborhood [84]	✓			✓
Extended Sorted Neighborhood [35,133]	✓		✓	
Hetero [100]		✓	✓	
Extended DNF BSL [101]		✓	✓	

load balancing as an optimization problem that minimizes not only the computational, but also the communication cost (e.g., network transfer time, local disk I/O time). The proposed solution provides strong theoretical guarantees for a performance close to the optimal one.

### 3.1 Discussion

Table 1 organizes the main schema-agnostic Blocking methods in a two-dimensional taxonomy that is formed by two criteria: (i) *Indexing Function Definition*, which determines whether learning is used to extract blocking keys from each entity description, and *Redundancy attitude*, which determines whether the outcome is a *redundancy-positive block collection*, where the more blocks two descriptions share, the more likely they are to be matching, or a *redundancy-neutral one* otherwise. We observe that most methods involve a non-learning functionality that produces redundancy-positive blocks. Among them, TB tries to maximize recall by assuming that duplicate entities share at least one common token in their values. Extensive experiments have shown that this assumption holds for KBs in the *center of the LOD cloud* [37, 57]. Yet, this coarse-grained approach typically leads to very low precision, since most of the pairs sharing a common word are non-matches. Attribute Clustering Blocking increases TB's precision by requiring that the common tokens of matching entities appear in attributes with similar values. Prefix-Infix(-Suffix) Blocking applies only to RDF data. However, it has been shown that both methods perform poorly when applied to KBs from the *periphery of the LOD cloud* [37, 57]. The reason is that they exclusively consider the noisy content of descriptions, disregarding the valuable evidence that is provided by contextual information, such as the neighboring descriptions, i.e., entities of different types connected via important relations. TYPiMatch also attempts to raise TB's precision, by categorizing the given entities into overlapping types, but its recall typically drops to a large extent, due to the noisy, schema-agnostic detection of entity types [141].

Overall, the schema-agnostic Blocking methods address both Volume and Variety of Big Data entities, consistently achieving high recall, due to redundancy. Their precision, though, is very low, due to the large portion of redundant and the superfluous comparisons in their overlapping blocks. We refer to [34, 35, 140] for a more detailed overview of Blocking methods.

## 338 4 BLOCK PROCESSING

339 This step receives as input a set of blocks  $\mathcal{B}$  and produces as output a new set of blocks  $\mathcal{B}'$  that has  
 340 similar recall, but significantly higher precision. This is achieved by discarding most superfluous  
 341 and redundant comparisons in  $\mathcal{B}$ . The relevant techniques operate at the coarse level of entire  
 342 blocks (Block Cleaning) or at the finer level of individual comparisons (Comparison Cleaning).

### 343 4.1 Block Cleaning

344 Methods of this type are *static*, i.e., independent of Matching, or *dynamic*, i.e., interwoven with it.

345 **Static methods.** The core assumption is that excessively large blocks (e.g., those corresponding to  
 346 stop-words) are dominated by unnecessary comparisons. In fact, the larger a block is, the less likely  
 347 it is to contain *unique duplicates*, i.e., matches that share no other block. Hence, they discard the  
 348 largest blocks, raising precision, without any significant impact on recall. To this end, *Block Purging*  
 349 sets an upper limit on the number of comparisons [136] or the block size [135]. *Block Filtering*  
 350 applies a limit to the blocks of every description, retaining it in  $r\%$  of its smallest blocks [139, 141].

351 More advanced methods, like a MapReduce-based blocking algorithm [119], learning-based (su-  
 352 pervised) method *Rollup Canopies* [157], and *Size-based Block Clustering* [65], split excessively large  
 353 blocks into smaller sub-blocks until they all satisfy the maximum block size limit. The last method  
 354 may merge back small blocks with similar blocking keys, in order to raise recall.

355 **Dynamic methods.** Assuming that Matching is performed by a *perfect oracle*, these methods  
 356 schedule the processing of blocks on-the-fly so as to maximize ER effectiveness and time efficiency.  
 357 For Dirty ER, *Iterative Blocking* [194] merges any new pair of matching descriptions,  $e_i$  and  $e_j$ , into  
 358 a new one,  $e_{i,j}$ , and replaces both  $e_i$  and  $e_j$  with  $e_{i,j}$  in all blocks that contain them. The already  
 359 processed blocks are reprocessed so that  $e_{i,j}$  is compared with all others; the new content in  $e_{i,j}$   
 360 may yield different similarity values that designate previously missed matches.

361 For Clean-Clean ER, *Block Scheduling* orders blocks in ascending order of comparisons [163],  
 362 or block size [136], so as to detect matches as early as possible. These matches are propagated  
 363 to subsequently processed blocks in order to reduce the superfluous comparisons. This yields a  
 364 block processing order with decreasing density of detected matches. Based on this observation,  
 365 *Block Pruning* [136] terminates the entire ER process as soon as the average number of executed  
 366 comparisons for detecting a new pair of duplicates drops below a predetermined threshold.

### 367 4.2 Comparison Cleaning

368 Most methods of this type operate on *redundancy-positive block collections*, where the more blocks  
 369 two descriptions share, the more likely they are to be matching. This characteristic allows for  
 370 weighting all pairwise comparisons in proportion to the matching likelihood of the corresponding  
 371 descriptions, a process that has been formalized by *Meta-blocking* [137].

372 Meta-blocking converts the input block collection  $\mathcal{B}$  into a *blocking graph*  $G_B$ , where nodes  
 373 correspond to descriptions and unique edges connect every pair of co-occurring descriptions. The  
 374 edges are weighted in proportion to the likelihood that the adjacent descriptions are matching.  
 375 Edges with low weights are pruned, as they probably correspond to superfluous comparisons. A  
 376 new block is then created for every retained edge, yielding the restructured block collection  $\mathcal{B}'$ .  
 377 In this process, various techniques can be used for weighting and pruning the graph edges [137].

378 For edge pruning, the following algorithms are available: *Weighted Edge Pruning* [137] removes  
 379 all edges that do not exceed the average edge weight; *Cardinality Edge Pruning* retains the globally  
 380  $K$  top weighted edges [137, 200]; *Weighted Node Pruning* (WNP) [137] and *BLAST* [161] retain in  
 381 each node neighborhood the descriptions that exceed a local threshold; *Cardinality Node Pruning*  
 382 (CNP) retains the top- $k$  weighted edges in each node neighborhood [137]; *Reciprocal WNP* and

Table 2. A Taxonomy of the Blocking Processing Methods Discussed in Section 4 (in the Order of Presentation)

	Granularity of Functionality		Matching awareness		Pruning Definition	
	Block Cleaning	Comparison Cleaning	dynamic	Static	non-learning	learning-based
Block Purging [135,136]	✓			✓	✓	
Block Filtering [139,141]	✓			✓	✓	
Rollup Canopies [157]	✓			✓		✓
Size-based Block Clustering [65]	✓			✓	✓	
Iterative Blocking [194]	✓		✓		✓	
Block Scheduling [136,163]	✓		✓		✓	
Block Pruning [136]	✓		✓		✓	
Weighted Edge Pruning [137]		✓		✓	✓	
Cardinality Edge Pruning [137,200]		✓		✓	✓	
(Reciprocal) Weighted Node Pruning [137,139]		✓		✓	✓	
BLAST [161]		✓		✓	✓	
(Reciprocal) Cardinality Node Pruning [137,139]		✓		✓	✓	
Disjunctive Blocking Graph [56]		✓		✓	✓	
Transitive LSH [167]		✓		✓	✓	
SPAN [160]		✓		✓	✓	
Comparison Propagation [136]		✓		✓	✓	
Supervised Meta-blocking [138]		✓		✓		✓
BLOSS [18]		✓		✓		✓

*CNP* [139] retain edges satisfying the pruning criteria in both adjacent node neighborhoods. Other methods perform edge pruning inside individual blocks [47], while *Disjunctive Blocking Graph* [56] associates every edge with multiple weights to express composite co-occurrence conditions.

On another line of research, *Transitive LSH* [167] converts LSH blocks into an unweighted blocking graph and applies a community detection algorithm, such as [40], while *SPAN* [160] uses matrix representations and operations to enhance the input block collection. The only approach that applies to any block collection  $\mathcal{B}$ , even one that is not redundancy-positive, is *Comparison Propagation* [136], which merely discards all redundant comparisons from  $\mathcal{B}$ .

**Learning-based methods.** *Supervised Meta-blocking* [138] casts edge pruning as a binary classification problem: every edge is annotated with a vector of schema-agnostic features, and is classified as likely match or unlikely match. *BLOSS* [18] further cuts down on the labeling effort, by selecting a very small training set that maintains high effectiveness.

**Parallelization.** Meta-blocking has been adapted to both multi-core [134] and MapReduce parallelization [55]. Regarding the latter, the *entity-based strategy* [55] aggregates for every description the bag of all description ids that co-occur with it in at least one block. Then, it estimates the edge weight that corresponds to each neighbor based on its frequency in the co-occurrence bag. An alternative approach is the *comparison-based strategy* [55]: the first pre-processing job enriches each block with the list of block ids associated with every description. This allows for computing the edge weights and discarding all redundant comparisons in the Map phase of the second job, while the superfluous comparisons are pruned in the ensuing Reduce phase. Both strategies rely on the load balancing algorithm *MaxBlock* [55] to avoid the underutilization of the available resources. BLAST is parallelized in [162], exploiting the broadcast join of Apache Spark for very high efficiency.

### 4.3 Discussion

Table 2 presents an overview of the Block Processing methods discussed above. The resulting taxonomy consists of three criteria: granularity of functionality, matching awareness (i.e., whether a

409 method is dynamic, depending on the outcomes of Entity Matching method, or static) and prun-  
 410 ing definition (i.e., whether the search space is reduced through a learning process that involves  
 411 labeled instances or not). Most Block Processing techniques involve a comparison-centric, static  
 412 and non-learning functionality that can be seamlessly combined with any Blocking technique.  
 413 Numerous studies have demonstrated that Block and Comparison Cleaning are indispensable for  
 414 schema-agnostic Blocking, raising precision by orders of magnitude, without hurting recall [136,  
 415 141, 161]. Multiple Block Cleaning methods can be part of the same end-to-end ER workflow, as  
 416 they are typically complementary; e.g., Block Purging is usually followed by Block Filtering [139].  
 417 Yet, at most one Comparison Cleaning method can be part of an ER workflow: applying it to  
 418 a redundancy-positive block collection removes its co-occurrence patterns and renders all other  
 419 techniques inapplicable. The top performer among non-learning techniques is BLAST [161], while  
 420 BLOSS performs better by labelling just ~50 instances [18]. We refer to [140] for a more detailed  
 421 overview of Block Processing techniques.

## 422 5 MATCHING

423 At the core of ER lies the *Matching* task, which receives as input a block collection and for each  
 424 pair of candidate matches that co-occur in a block, it decides if they refer to the same real-world  
 425 entity.

### 426 5.1 Preliminaries

427 The matching decision is typically made by a match function  $M$ , which maps each pair of entity  
 428 descriptions  $(e_i, e_j)$  to  $\{true, false\}$ , with  $M(e_i, e_j) = true$  meaning that  $e_i$  and  $e_j$  are matches, and  
 429  $M(e_i, e_j) = false$  meaning that  $e_i$  and  $e_j$  are not matches.

430 In its simplest form,  $M$  is defined via a similarity function  $sim$ , measuring how similar two en-  
 431 tities are to each other, according to certain comparison attributes.  $sim$  can consist of an *atomic*  
 432 similarity measure, like Jaccard similarity, or a *composite* one, e.g., a linear combination of sev-  
 433 eral atomic similarity functions on different attributes of a description. To specify an equivalence  
 434 relation among entity descriptions, we need to consider a similarity measure satisfying the non-  
 435 negativity, identity, symmetry, and triangle inequality properties [198], i.e., a similarity *metric*.  
 436 Given a similarity threshold  $\theta$ , a simple matching function can be defined as

$$M(e_i, e_j) = \begin{cases} true, & \text{if } sim(e_i, e_j) \geq \theta, \\ false, & \text{otherwise.} \end{cases}$$

437 In more complex ER pipelines, such as when matching rules are manually provided, or learned  
 438 from training data, the matching function  $M$  can be defined as a complex function involving several  
 439 matching conditions. For instance, two person descriptions match if their SSN is identical, or if  
 440 their date of birth, zip code, and last names are identical, or if their e-mail addresses are identical.

441 Finding a similarity metric which can perfectly distinguish all matches from non-matches using  
 442 simple pairwise comparisons on the attribute values of two descriptions is practically impossi-  
 443 ble. In particular, similarity metrics are too restrictive to identify nearly similar matches. Thus, in  
 444 reality, we seek similarity functions that will be only good enough, i.e., minimize the number of  
 445 misclassified pairs, and rely on collective ER approaches to propagate the similarity of the entity  
 446 neighbors of two descriptions to the similarity of those descriptions. In this inherently iterative  
 447 process, the employed match function is based on a similarity that dynamically changes from it-  
 448 eration to iteration, and its results may include a third state, the *uncertain* one. Specifically, given

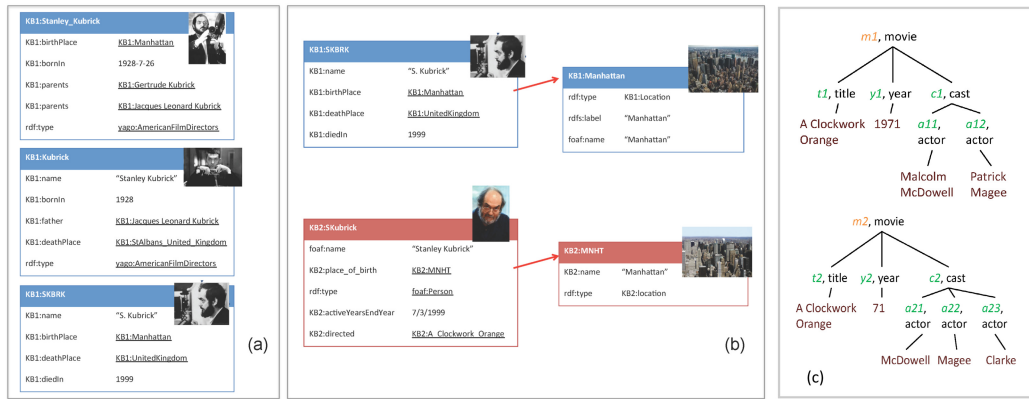


Fig. 4. (a) A merging-based collective ER example and (b) a relationship-based collective ER example. (c) Two different descriptions of the movie *A Clockwork Orange* and its cast in XML.

two similarity thresholds  $\theta$  and  $\theta'$ , with  $\theta' < \theta$ , the match function at iteration  $n$ ,  $M^n$ , is given by 449

$$M^n(e_i, e_j) = \begin{cases} \text{true, if } sim^{n-1}(e_i, e_j) \geq \theta, \\ \text{false, if } sim^{n-1}(e_i, e_j) \leq \theta', \\ \text{uncertain, otherwise.} \end{cases}$$

Based on the characteristics of the entity collections (e.g., structuredness, domain, size), the nature of comparisons (attribute-based or collective), as well as the availability of known, pre-labeled matching pairs, different methodologies can be followed to identify an appropriate similarity function and thus, a fitting match function. In what follows, we explore alternative methodologies for the matching task and discuss the cases in which those methodologies are more suited. 450 451 452 453 454

## 5.2 Collective Methods 455

To minimize the number of missed matches, commonly corresponding to nearly similar matches, a collective ER process can jointly discover matches of inter-related descriptions. This is an inherently iterative process that entails additional processing cost. We distinguish between *merging-* and *relationship-based* collective ER approaches. In the former, new matches can be identified by exploiting the merging of the previously found matches, while in the latter, iterations rely on the similarity evidence provided by descriptions being structurally related in the original entity graph. 456 457 458 459 460 461

*Example 5.1.* Consider the descriptions in Figure 4 (a), which stem from the knowledge base KB1. They all refer to the person, Stanley Kubrick. Initially, it is difficult to match KB1:SKBRK with any other description, since many people named Kubrick may have been born in Manhattan, or died in the UK, respectively. However, it is quite safe to match the first two descriptions (KB1:Stanley\_Kubrick and KB1:Kubrick). By merging the first two descriptions, e.g., using the union of their attribute-value pairs, it becomes easier to identify that the last description (KB1:SKBRK) also refers to the same person, based on the name and the places of birth and death. Consider now the descriptions in Figure 4(b), which stem from the knowledge bases KB1 and KB2. The descriptions on the left (KB1:SKBRK and KB2:SKubrick) represent Stanley Kubrick, while the descriptions on the right (KB1:Manhattan and KB2:MNHT) represent Manhattan, where Kubrick was born. Initially, it is difficult to identify the match between the descriptions on the left, based only on the common year of death and last name. However, it is quite straightforward to identify the match between the descriptions of Manhattan, on the right. Having identified this match, a relationship-based collective ER algorithm would re-consider matching KB1:SKBRK to KB2:SKubrick, since these 462 463 464 465 466 467 468 469 470 471 472 473 474 475

476 descriptions are additionally related, with the same kind of relationship (birth place), to the descrip-  
 477 tions of Manhattan that were previously matched. Therefore, a relationship-based ER algorithm  
 478 would identify this new match in a second iteration.

479 Note that the structuredness of the input entity collection to be resolved is a key factor for  
 480 the nature of collective approaches. Merging-based methods are typically schema-aware, since  
 481 structured data make the process of merging easier. On the other hand, collective methods dealing  
 482 with semi-structured data are typically relationship-based, since merging would require deciding  
 483 not only which values are correct for a given attribute, but also which values are available for  
 484 similar attributes and can be used to merge two descriptions.

485 *5.2.1 Schema-Aware Methods.* In *merging-based collective ER*, the matching decision between  
 486 two descriptions triggers a merge operation, which transforms the initial entity collection by  
 487 adding the new, merged description and potentially removing the two initial descriptions. This  
 488 change also triggers more updates in the matching decisions, since the new, merged description  
 489 needs to be compared to the other descriptions of the collection. Intuitively, the final result of  
 490 merging-based collective ER is a new entity collection, which is the result of merging all the  
 491 matches found in the initial collection. In other words, there should be a one-to-one correspon-  
 492 dence between the descriptions in the resolution results and the actual real-world entities from  
 493 the input entity collection.

494 Considering the functions of matching  $M$  and merging  $\mu$  as black boxes, *Swoosh* [15] is a family  
 495 of merging-based collective ER strategies that minimize the number of invocations to these poten-  
 496 tially expensive black boxes; *D-Swoosh* [14] introduces a family of algorithms that distribute the  
 497 workload of merging-based ER across multiple processors. Both works rely on the following set  
 498 of *ICAR* properties, that, when satisfied by  $M$  and  $\mu$ , lead to higher efficiency:

- 499 – *Idempotence*:  $\forall e_i, M(e_i, e_i) = true$  and  $\mu(e_i, e_i) = e_i$ .
- 500 – *Commutativity*:  $\forall e_i, e_j, M(e_i, e_j) = true \Leftrightarrow M(e_j, e_i) = true$  and  $\mu(e_i, e_j) = \mu(e_j, e_i)$ .
- 501 – *Associativity*:  $\forall e_i, e_j, e_k$ , if both  $\mu(e_i, \mu(e_j, e_k))$  and  $\mu(\mu(e_i, e_j), e_k)$  exist,  $\mu(e_i, \mu(e_j, e_k)) =$   
 502  $\mu(\mu(e_i, e_j), e_k)$ .
- 503 – *Representativity*: If  $e_k = \mu(e_i, e_j)$ , then for any  $e_l$  such that  $M(e_i, e_l) = true, M(e_k, e_l) = true$ .

504 Regarding the match function, idempotence and commutativity have been already discussed  
 505 in Section 5.1, as reflexivity and symmetry, respectively, while representativity extends transi-  
 506 tivity, by also including the merge function. Note that if associativity does not hold, it becomes  
 507 harder to interpret a merged description, since it depends on the order in which the source de-  
 508 scriptions were merged.

509 *R-Swoosh* [15] exploits the *ICAR* properties as follows. A set  $\mathcal{E}$  of entity descriptions is initialized  
 510 to contain all the input descriptions. Then, in each iteration, a description  $e$  is removed from  $\mathcal{E}$   
 511 and compared to each description  $e'$  of the, initially empty, set  $\mathcal{E}'$ . If  $e$  and  $e'$  are found to match,  
 512 then they are removed from  $\mathcal{E}$  and  $\mathcal{E}'$ , respectively, and the result of their merging is placed into  
 513  $\mathcal{E}$  (exploiting representativity). If there is no description  $e'$  matching with  $e$ , then  $e$  is placed in  $\mathcal{E}'$ .  
 514 This process continues until  $\mathcal{E}$  becomes empty, i.e., there are no more matches to be found.

515 In *relationship-based collective ER*, the matching decision between two descriptions triggers dis-  
 516 covering new candidate pairs for resolution, or re-considering pairs already compared; matched  
 517 descriptions may be related to other descriptions, which are now more likely to match to  
 518 each other.

519 To illustrate the relationships between the descriptions of an entity collection  $\mathcal{E}$ , usually, an *en-*  
 520 *tity graph*  $G_{\mathcal{E}} = (V, E)$  is used, in which nodes,  $V \subseteq \mathcal{E}$ , represent entity descriptions and edges,  
 521  $E$ , reflect the relationships between the nodes. For example, such a match function could be

of the form

522

$$M(e_i, e_j) = \begin{cases} true, & \text{if } sim(nbr(e_i), nbr(e_j)) \geq \theta \\ false, & \text{else,} \end{cases}$$

where  $sim$  can be a relational similarity function and  $\theta$  is a threshold value. Intuitively, the neighborhood  $nbr(e)$  of a node  $e$  can be the set of all the nodes connected to  $e$ , i.e.,  $nbr(e) = \{e_j | (e, e_j) \in E\}$ , or the set of edges containing  $e$ , i.e.,  $nbr(e) = \{(e, e_j) | (e, e_j) \in E\}$ .

*Collective ER* [16] employs an entity graph, following the intuition that two nodes are more likely to match, if their edges connect to nodes corresponding to the same entity. To capture this iterative intuition, hierarchical agglomerative clustering is performed, where, at each iteration, the two most similar clusters are merged, until the similarity of the most similar clusters is below a threshold. When two clusters are merged, the similarities of their related clusters, i.e., the clusters corresponding to descriptions related to the descriptions in the merged cluster, are updated. To avoid comparing all the pairs of input descriptions, Canopy Clustering [118] is initially applied.

*Hybrid Collective ER* [48] is based on both partial merging results and relations between descriptions. It constructs a dependency graph, where every node represents the similarity between a pair of entity descriptions and every edge represents the dependency between the matching decisions of two nodes. If the similarity of a pair of descriptions changes, the neighbors of this pair might need a similarity re-computation. The dependencies between the matching decisions are distinguished between Boolean and real-valued. The former suggest that the similarity of a node depends only on whether the descriptions of its neighbor node match or not, while in real-valued dependencies, the similarity of a node depends on the similarity of the descriptions of its neighbor node. Boolean dependencies are further divided into strong (if a node corresponds to a match, its neighbor pair should also be a match), and weak (if a node corresponds to a match, the similarity of its neighbor pair is increased). Initially, all nodes are added to a priority queue. On each iteration, a node is removed from the queue and if the similarity of the node is above a threshold, its descriptions are merged, aggregating their attribute values, to enable further matching decisions; if the similarity value of this node has increased, its neighbor nodes are added to the priority queue. This iterative process continues until the priority queue becomes empty.

**5.2.2 Schema-Agnostic Methods.** *Collective ER for tree (XML) data* is studied in [190]. Entity descriptions correspond to XML elements composed of text data or other XML elements, and domain experts specify which XML elements are match candidates, thus, initializing a priority queue of comparisons. Entity dependency takes the following form in this case: an XML element  $c$  depends on another XML element  $c'$ , if  $c'$  is a part of the description of  $c$ . Consequently, identifying the matches of  $c$  is not independent of identifying the matches of  $c'$ . Even if two XML elements are initially considered to be non-matches, they are compared again, if their related elements are marked as matches. A similar approach is based on the intuition that the similarity of two elements reflects the similarity of their data, as well as the similarity of their children [189]. Following a top-down traversal of XML data, the DELPHI containment metric [6] is used to compare two elements.

*Example 5.2.* Figure 4(c) shows two different descriptions of the movie *A Clockwork Orange* in XML. This representation means that the element *movie* consists of the elements *title*, *year*, and *cast*, with the last one further consists of *actor* elements. To identify that the two XML descriptions represent the same movie, we can start by examining the cast of the movies. After we identify that actors  $a_{11}$  and  $a_{21}$  represent the same person, Malcolm McDowell, the chances that the movies  $m_1$  and  $m_2$  match are increased. They are further increased when we find that actors  $a_{12}$  and  $a_{22}$  also match, representing Patrick Magee. The same matching process over all the sub-elements of  $m_1$  and  $m_2$  will finally lead us to identify that  $m_1$  and  $m_2$  match.



566 *SiGMA* [111] selects as seed matches the pairs that have identical entity names. Then, it propa-  
 567 gates the matching decisions on the compatible neighbors of existing matches. Unique Mapping  
 568 Clustering is applied for detecting duplicates. For every new matched pair, the similarities of the  
 569 neighbors are recomputed and their position in the priority queue is updated.

570 *LINDA* [21] follows a very similar approach, which differs from *SiGMA* mainly in the similarity  
 571 functions and the lack of a manual relation alignment. *LINDA* relies on the edit distance of the  
 572 relation names used in the two KBs to determine if they are equivalent or not. This alignment  
 573 method makes a strong assumption that descriptions in KBs use meaningful names for relations  
 574 and similar names for equivalent relations, which is often not true in the Web of Data. Rather than  
 575 using a similarity threshold, the resolution process in *LINDA* terminates when the priority queue  
 576 is empty, or after performing a predetermined number of iterations.

577 *RiMOM-IM* [114, 159] initially considers as matches entities placed in blocks of size 2. It also  
 578 uses a heuristic called “one-left object”: if two matched descriptions  $e_1, e'_1$  are connected via aligned  
 579 relations  $r$  and  $r'$ , and all their entity neighbors via  $r$  and  $r'$ , except  $e_2$  and  $e'_2$ , have been matched,  
 580 then  $e_2, e'_2$  are also considered matches. Similar to *SiGMA*, *RiMOM-IM* employs a complex similarity  
 581 score, which requires the alignment of relations among the KBs.

582 *PARIS* [169] uses a probabilistic model to identify matching evidence, based on previous matches  
 583 and the functional nature of entity relations. A relation is considered to be functional if, for a given  
 584 source entity, there is only one destination entity (e.g., *wasBornIn*). The basic matching idea is  
 585 that if  $r(x, y)$  is a function in one KB and  $r(x, y')$  is a function in another KB, then  $y$  and  $y'$  are  
 586 considered to be matches. The *functionality*, i.e., degree by which a relation is close to being a  
 587 function, and the alignment of relations along with previous matching decisions determine the  
 588 decisions in subsequent iterations. The functionality of each relation is computed at the beginning  
 589 of the algorithm and remains unchanged. Initially, instances with identical values (for all attributes)  
 590 are considered matches and based on those matches, an alignment of relations takes place. In every  
 591 iteration, instances are compared based on the newly aligned relations, and this process continues  
 592 until convergence. In the last step, an alignment of classes (i.e., entity types) also takes place.

593 On another line of research, *MinoanER* [56] executes a non-iterative process that involves four  
 594 matching rules. First, it identifies matches based on their name (rule R1). This is a very effective  
 595 and efficient method that can be applied to all descriptions, regardless of their values or neighbor  
 596 similarity, by automatically specifying distinctive names of entities based on data statistics. Then,  
 597 the value similarity is exploited to find matches with many common and infrequent tokens, i.e.,  
 598 strongly similar matches (rule R2). When value similarity is not high, nearly similar matches are  
 599 identified based on both value and neighbors similarity using a threshold-free rank aggregation  
 600 function (rule R3). Finally, reciprocal evidence of matching is exploited as a verification of the  
 601 returned results: only entities mutually ranked in the top matching candidate positions of their  
 602 unified ranking lists are considered as matches (rule R4).

### 603 5.3 Learning-Based Methods

604 The first probabilistic model for ER [63] used attribute similarities as the dimensions of comparison  
 605 vectors, each representing the probability that a pair of descriptions match. Following the same  
 606 conceptual model, a large number of works try to automate the process of learning such probabili-  
 607 ties based on manually or automatically generated, or even pre-existing training data. Next, we  
 608 explore different ways of generating and exploiting training data.

609 **Supervised Learning.** *Adaptive Matching* [41] learns from the training data a composite function  
 610 that combines many attribute similarity measures. Similarly, *MARLIN* [20] uses labeled data at  
 611 two levels. First, it can utilize trainable string similarity/distance measures, such as learnable edit

distance, adapting textual similarity computations to specific attributes. Second, it uses labeled data to train a classifier that distinguishes pairs between matches and non-matches, using textual similarity values for different attributes as features.

*Gradient-Based Matching* [150] proposes a model that can adjust its structure and parameters based on aggregate similarity scores coming from individual similarity functions on different attributes. Its design allows for locating which similarity functions and attributes are more significant to correctly classify pairs. For its training, it employs a performance index that helps to separate descriptions that have already been matched from those that have not been matched as yet.

*BN-Based Collective ER* [89] adapts a relationship-based collective ER approach (similar to [48]) to a supervised learning setting. A Bayesian network is used to capture cause-effect relationships, which are modeled as directed acyclic graphs, and to compute matching probabilities. The lexical similarity in the attribute values of the descriptions as well as their links to existing matches constitute positive matching evidence, which incrementally updates the Bayesian network nodes, similar to the incremental updates that take place in the graph-based dependency model of [48].

*GenLink* [91] is a supervised, genetic programming algorithm for learning expressive linkage rules, i.e., functions that assign similarity values to pairs of descriptions. GenLink generates linkage rules that select the important attributes for comparing two descriptions, normalize their attribute values before similarity computations, choose appropriate similarity measures and thresholds, and combine the results of multiple comparisons using linear as well as non-linear aggregation functions. It has been incorporated into the Silk Link Discovery Framework [180] (see Section 9.5).

**Weakly Supervised Learning.** Arguably, the biggest limitation of supervised approaches is the need for a labeled dataset, based on which the underlying machine learning algorithm will learn how to classify new instances. Methods of this category reduce the cost of acquiring such a dataset.

A *transfer learning* approach is proposed in [173] with the aim of adapting and reusing labeled data from a related dataset. The idea is to use a standardized feature space in which the entity embeddings of the reused and the targeted dataset will be transferred. This way, existing labeled data from another dataset can be used to train a classifier that can work with the target dataset, even if there are no explicitly labeled data for the target dataset. A similar transfer learning approach is also followed in [152] to infer equivalence links in a linked data setting.

*Snorkel* [149] is a generic tool that can be used to generate training data for a broader range of problems than ER. It relies on user-provided heuristic rules (e.g., several matching functions) to label some user-provided data and evaluate this labeling using a small pre-labeled dataset. Instead of attribute weighting, Snorkel tries to learn the importance of the provided matching functions. This approach of weighting matching rules, instead of features, resembles and complements existing works in ER. For example, the goal in [184] is to identify which similarity measure can maximize a specific objective function for an ER task, given a set of positive and negative examples. Those examples can be generated manually one-by-one, or by leveraging tools like Snorkel.

**Unsupervised Learning.** *Unsupervised Ensemble Learning* [94] generates an ensemble of automatic self-learning models that use different similarity measures. To enhance the automatic self-learning process, it incorporates attribute weighting into the automatic seed selection for each of the self-learning models. To ensure that there is high diversity among the selected self-learning models, it utilizes an unsupervised diversity measure. Based on it, the self-learning models with high contribution ratios are kept, while the ones with poor accuracy are discarded.

Rather than relying on domain expertise or manually labeled samples, the unsupervised ER system presented in [102] automatically generates its own heuristic training set. As positive examples are considered the pair of descriptions with very high Jaccard similarity of the token sets in their

659 attribute values. In the context of Clean-Clean ER, having generated the positive example ( $e_1, e_2$ ),  
660 where  $e_1$  belongs to entity collection  $\mathcal{E}_1$  and  $e_2$  to  $\mathcal{E}_2$ , for every other positive example ( $e_3, e_4$ ),  
661 where  $e_3 \in \mathcal{E}_1$  and  $e_4 \in \mathcal{E}_2$ , it further infers the negative examples ( $e_1, e_4$ ) and ( $e_3, e_2$ ). The result-  
662 ing training set is first used by the system for Schema Matching to align the attributes in the input  
663 datasets. The attribute alignment and the training sets are then used to simultaneously learn two  
664 functions—one for Blocking and the other for Matching.

#### 665 5.4 Parallel Methods

666 We now discuss works that are able to leverage massive parallelization frameworks.

667 A framework for scaling collective ER [16] to large datasets is proposed in [148], assuming a  
668 black-box ER algorithm. To achieve high scalability, it runs multiple instances of the ER algorithm  
669 in small subsets of the entity descriptions. An initial block collection is constructed based on the  
670 similarity of the descriptions using Canopy Clustering [118]. Each block is then extended by taking  
671 its *boundary* with respect to entity relationships. Next, a simple message-passing algorithm is run,  
672 to ensure that the match decisions within a block, which might influence the match decisions in  
673 other blocks, are propagated to those other blocks. This algorithm retains a list of active blocks,  
674 which initially contains all blocks. The black-box ER algorithm is run locally, for each active block,  
675 and the newly identified matches are added in the result set. All the blocks with a description of  
676 the newly identified matches are set as active. This iterative algorithm terminates when the list of  
677 active blocks becomes empty.

678 *LINDA* [21] scales out using MapReduce. The pairs of descriptions are sorted in descending order  
679 of similarity and stored in a priority queue. Each cluster node holds (i) a partition of this priority  
680 queue, and (ii) the corresponding part of the entity graph, which contains the descriptions in the  
681 local priority queue partition along with their neighbors. The iteration step of the algorithm is that,  
682 by default, the first pair in the priority queue is considered to be a match and is then removed from  
683 the queue and added to the known matches. This knowledge triggers similarity re-computations,  
684 which affect the priority queue by (i) enlarging it, when the neighbors of the new match are added  
685 again to the queue, (ii) re-ordering it, when the neighbors of the identified match move higher in  
686 the rank, or (iii) shrinking it, after applying transitivity and the constraint for a unique match per  
687 KB. The algorithm stops when the priority queue is empty, or after a specific number of iterations.

688 Finally, *Minoan-ER* [56] runs on top of Apache Spark. To minimize its overall runtime, it ap-  
689 plies Name Blocking, while extracting the top similar neighbors per entity and running Token  
690 Blocking. Then, it synchronizes the results of the last two processes: it combines the value simi-  
691 larities computed by Token Blocking with the top neighbors per entity to estimate the neighbor  
692 similarities for all entity pairs with neighbors co-occurring in at least one block. Matching rule  
693 R1 (finding matches based on their name) starts right after Name Blocking, R2 (finding strongly  
694 similar matches) after H1 and Token Blocking, R3 (finding nearly similar matches) after R2 and  
695 the computation of neighbor similarities, while R4 (the reciprocity filter) runs last, providing the  
696 final, filtered set of matches. During the execution of every rule, each Spark worker contains only  
697 the partial information of the blocking graph that is necessary to find the match of a specific node.

#### 698 5.5 Discussion

699 Table 3 presents an overview of the Matching methods discussed in this section. They are or-  
700 ganized according to schema-awareness (schema-aware or schema-agnostic), nature of compar-  
701 isons (attribute-based or collective), and algorithmic foundations (non-learning or learning-based).  
702 Collective methods are further refined as merging-based (MB) or relationship-based (RB), and  
703 learning-based methods as supervised (S), weakly supervised (WS), and unsupervised (U).

Table 3. Taxonomy of the Matching Methods Discussed in Section 5

	Schema awareness		Nature of comparisons		Algorithmic foundations	
	Schema-aware	Schema-agnostic	Attribute-based	Collective	Learning-based	Non-learning
Swoosh [15]	✓			MB		✓
D-Swoosh [14]	✓			MB		✓
Collective ER [16]	✓			RB		✓
Hybrid Collective ER [48]	✓			MB,RB		✓
Collective ER for XML [190]		✓		RB		✓
SIGMa [111]		✓		RB		✓
LINDA [21]		✓		RB		✓
RiMOM [114, 159]		✓		RB		✓
PARIS [169]		✓		RB		✓
MinoanER [56]		✓		RB		✓
Adaptive Matching [41]	✓		✓		S	
MARLIN [20]	✓		✓		S	
Gradient-based Matching [150]	✓		✓		S	
BN-based Collective ER [89]		✓		RB	S	
GenLink [91]		✓	✓		S	
Transfer learning [173]	✓		✓		WS	
Transfer learning for RDF [152]		✓		RB	WS	
Unsupervised ensemble [94]	✓		✓		U	
Unsupervised ER for RDF [102]	✓		✓		U	
Large-scale Collective ER [148]	✓			RB		✓

MB stands for Merging-based, RB for Relationship-based, S for Supervised, WS for Weakly Supervised, and U for Unsupervised Learning.

We observe that all schema-agnostic methods that have been proposed are collective, and more specifically, relationship-based. This happens because, unlike the schema-aware methods, the schema-agnostic ones cannot rely on attribute-level similarities for attributes that are not known in advance, or it is not known if they are actually used by the descriptions. Hence, those methods propagate the information provided by entity neighbors as matching evidence whenever possible. Consequently, as a rule of thumb that depends on the nature of the input data, we recommend merging-based collective ER methods, which are schema-aware, for data coming from a single dirty entity collection (e.g., for the deduplication of a dirty customer data base) and relationship-based collective ER methods, which are schema-agnostic, for data coming from multiple, curated entity collections (e.g., for finding equivalent descriptions among two or more Web KBs).

Note that the learning-based methods can be seen as *attribute-based*, since they essentially try to learn the probability that two descriptions match based on previous examples of similar pairs, or *collective*, since their models are trained on sets of pairs, or even on vectorial representations of entity descriptions, or the words used in the values of those descriptions. For completeness, Table 3 classifies them as attribute-based, following the traditional learning approach, because their collective nature cannot be easily labeled as merging-based or relationship-based. We believe that the learning-based methods are gaining ground as new and more effective ways to represent individual or groups of entity descriptions appear (see Section 9.1). The emergence of weakly supervised and transfer-learning methods seem to alleviate the problem of generating a labeled set for training data. Therefore, when labeled examples are available (e.g., in transfer learning), or

724 are easy to generate using existing tools (e.g., [149]), and the test data are not expected to deviate  
725 considerably from the training data, those methods seem to be the most promising ones. Before  
726 choosing learning-based or non-learning methods, one should also consider the desired frequency  
727 of re-training a new classification model, the memory footprint of each method (i.e., whether the  
728 whole model needs to reside in memory or not), and the time needed for training and classification.

729 In general, recent studies [52, 104, 122] show that the learning-based techniques achieve higher  
730 accuracy than the rule-based ones that are used in several practical scenarios. Yet, despite some past  
731 efforts (e.g., [90, 105, 106]), we notice the lack of a systematic benchmarking of matching methods.  
732 A comprehensive benchmark should evaluate effectiveness (i.e., quality of the output matches),  
733 time and space efficiency (i.e., the time required for pre-processing, training, and matching, the  
734 memory and disk space required by each method), and scalability (i.e., using the same computa-  
735 tional and storage resources, what is the data limit that each method can handle).

## 736 6 CLUSTERING METHODS

737 Typically, clustering constitutes the final task in the end-to-end ER workflow, following Matching.  
738 Its input comprises the *similarity graph*, where the nodes correspond to the descriptions and each  
739 edge connects a pair of descriptions that were compared during Matching; the edge weights, typ-  
740 ically in  $[0, 1]$ , are analogous to the matching likelihood of the adjacent descriptions. Clustering  
741 aims to infer more edges from indirect matching relations, while discarding edges that are un-  
742 likely to connect duplicates in favor of edges with higher weights. Hence, its end result is a set of  
743 *entity clusters*, each of which comprises all descriptions that correspond to the same, distinct real-  
744 world object.

745 In the simplest case, *Connected Components* [80, 153] is applied to compute the transitive closure  
746 of the detected matches. This naive approach increases recall, but is rather sensitive to noise.  
747 False positives have a significant impact on precision, leading to entity clusters that are dominated  
748 by non-matching descriptions. For this reason, more advanced clustering techniques have been  
749 proposed to leverage the weighted edges in the similarity graph. In general, these techniques are  
750 distinguished into three categories, according to the type of ER task at hand:

751 (1) For Clean-Clean ER, clustering typically relies on the one-to-one correspondence between  
752 the input data sources. The most popular technique is *Unique Mapping Clustering* [21, 111], which  
753 first sorts all edges in decreasing weight. At each iteration, the top edge is considered a match, if  
754 none of the adjacent descriptions has already been matched. The process ends when the top edge  
755 has a similarity lower than a threshold  $t$ . Essentially, this approach provides an efficient solution  
756 to the *Stable Marriage* problem for unequal sets [120], given that Clean-Clean ER forms a (usually  
757 unbalanced) bipartite similarity graph. The *Hungarian algorithm* is also applicable, though at a  
758 much higher computational cost, unless an approximation is used, as in [46, 108].

759 (2) For Dirty ER, the core characteristic of clustering algorithms is that they produce a set of  
760 disjoint entity clusters without requiring as input the number of clusters or any labeled dataset  
761 for training [80]. *Center Clustering* [82] iterates once over all edges and creates clusters around  
762 nodes that are selected as centers. Its functionality is enhanced by *Merge-Center Clustering* [81],  
763 which merges together clusters with centers similar to the same node. *Star Clustering* [10] be-  
764 gins with sorting all similarity graph nodes in descending order of degree. Then, the top node  
765 becomes the center of a cluster that includes all its direct neighbors. The same process is repeat-  
766 edly applied to the remaining nodes, until all nodes belong to a cluster. The resulting clusters are  
767 overlapping, unless post-processing assigns each node to a single cluster. *Ricochet Clustering* [195]  
768 comprises a family of techniques based on two alternating stages: the first one determines the cen-  
769 ters of clusters (like Star Clustering), while the second one (re-)assigns nodes to cluster centers (like  
770  $k$ -means).

Other techniques focus on the relative strength of the links inside and across clusters, i.e., the *intra-* and *inter-cluster* edges. *Markov Clustering* [175] uses random walks to strengthen the intra-cluster edges, while weakening the inter-cluster ones. *Cut clustering* [66] iteratively identifies the minimum cut of maximum flow paths from a node to an artificial sink node. This way, it detects small inter-cluster cuts, while strengthening intra-cluster links. *Correlation Clustering* [12] solves an optimization task, where the goal is to maximize the sum of the intra-cluster edges, while minimizing the sum of the inter-cluster ones. This is an NP-hard problem that is typically solved through approximations, such as *Clustering Aggregation* [73] and *Restricted Correlation Clustering* [109]. The latter is a semi-supervised approach that leverages a small labeled dataset, which is carefully selected via an efficient sampling procedure based on LSH.

(3) For Multi-source ER [153], we can use most algorithms for Dirty ER, but the multitude of input entity collections calls for specialized clustering methods. *SplitMerge* [126] applies Connected Components clustering and cleans the resulting clusters by iteratively removing entities with low similarity to other cluster members. Then, it merges similar clusters that are likely to correspond to the same real-world entity. *CLIP* [155] assumes duplicate-free entity collections as input. First, it computes the transitive closure of the strong links, i.e., the edges that correspond to the maximum weight per source (entity collection) for both adjacent nodes. The remaining graph is cleaned from the weak links, i.e., the edges that do not correspond to the maximum weight per source for neither adjacent node. Finally, the transitive closure is computed and its clusters are processed to ensure that they contain at most one description per source.

**Discussion.** The relative performance of Dirty ER methods has been experimentally evaluated in [80]. As expected, Connected Components exhibits the worst accuracy. Ricochet Clustering performs well only over entity collections with uniformly distributed duplicates, while Markov Clustering consistently achieves top performance. Surprisingly enough, the highly scalable, single-pass algorithms Center and Merge-Center clustering provide comparable, if not better, results than more complex techniques, like Cut and Correlation Clustering.

The relative performance of Multi-source ER algorithms is examined in [153, 154], using their parallelization in Apache Flink. The experiments show that SplitMerge and CLIP achieve the top performance, with the latter providing a better balance between effectiveness and time efficiency.

## 7 BUDGET-AWARE ER

Unlike the budget-agnostic methods presented above, budget-aware ER provides the best possible *partial solution*, when the response time or the available computational resources are constrained. It is driven by a *pay-as-you-go* paradigm that sacrifices the completeness of results, when the number of data sources or the amount of data to be processed is ever increasing. For example, the number of high-quality HTML tables on the Web is in the hundreds of millions, while the Google search system alone has indexed ~26 billion datasets [75]. This unprecedented volume of data can only be resolved progressively, using matching pairs from former iterations to generate more accurate candidate pairs in the latter iterations as long as the allocated budget is not exhausted.

Typically, budget-aware methods rely on blocking as a pre-processing task that identifies similar entity descriptions. They differ, though, on how they leverage the resulting blocks in the Planning step (see Figure 2(b)). Four categories of granularity functionality are defined [163] as follows:

- (1) *Block-centric methods* produce a list of blocks that are sorted in descending order of the likelihood that they include duplicates among their descriptions. All the comparisons inside each block are generated iteratively, one block at a time, following that ordered list.

- 816 (2) *Comparison-centric methods* provide a list of description pairs sorted in descending order  
 817 of matching likelihood. These pairs of descriptions are emitted iteratively, one at a time,  
 818 following that ordered list.
- 819 (3) *Entity-centric methods* provide a list of descriptions sorted in descending order of dupli-  
 820 cation likelihood. All comparisons of every description are generated iteratively, one de-  
 821 scription at a time, following that ordered list.
- 822 (4) The *hybrid methods* combine characteristics from two or all of the previous categories.

823 Depending on their blocking keys, budget-aware methods are further classified into [163] the  
 824 following:

- 825 (1) *Sort-based methods*, which rely on the similarity of blocking keys. They produce a list of  
 826 descriptions by sorting them alphabetically, according to their blocking keys, and assume  
 827 that the matching likelihood of two descriptions is analogous to their proximity after  
 828 sorting.
- 829 (2) *Hash-based methods*, which consider identical blocking keys and typically assume  
 830 redundancy-positive blocks, i.e., the similarity of two descriptions is proportional to their  
 831 common blocks.

832 In the sequel, we examine separately the schema-aware and the schema-agnostic methods.

### 833 7.1 Schema-Aware Methods

834 The budget-aware methods that are suitable for structured data rely on schema knowledge. This  
 835 means that their performance depends heavily on the attribute(s) that provide the schema-aware  
 836 blocking keys they leverage, typically requiring domain experts to fine-tune them.

837 The core comparison-centric method is *Progressive Sorted Neighborhood* (PSN) [193]. Based on  
 838 Sorted Neighborhood [84], it associates every description with a schema-aware blocking key. Then,  
 839 it produces a *sorted list of descriptions* by ordering all blocking keys alphabetically. Comparisons are  
 840 progressively defined through a sliding window,  $w$ , whose size is *iteratively incremented*: initially,  
 841 all descriptions in consecutive positions ( $w = 1$ ) are compared, starting from the top of the list;  
 842 then, all descriptions at distance  $w = 2$  are compared and so on, until termination.

843 The above approach produces a *static* list of comparisons, which remains immutable, regardless  
 844 of the duplicates that are identified. As a result, PSN cannot react to the skewed distribution of  
 845 duplicates. To ameliorate this issue, a *dynamic* version of the algorithm was proposed in [143]. Its  
 846 functionality is integrated with Matching to adjust the processing order of comparisons on-the-fly.  
 847 Arranging the sorted descriptions in a two-dimensional array  $A$ , if position  $A(i, j)$  corresponds to  
 848 a duplicate, the processing moves on to check positions  $A(i + 1, j)$  and  $A(i, j + 1)$ .

849 The same principle lies at the core of the dynamic, block-centric method *Progressive Block-*  
 850 *ing* [143]. Initially, a set of blocks is created and its elements are arranged in a two-dimensional  
 851 array  $A$ . Then, all comparisons are executed inside every block, measuring the number of dupli-  
 852 cates per block. Starting from the block with the highest density of duplicates in position  $A(i, j)$ ,  
 853 its descriptions are compared with those in the blocks  $A(i + 1, j)$  and  $A(i, j + 1)$  in order to identify  
 854 more matches.

855 A static, block-centric method is the *Hierarchy of Record Partitions* (HRP) [193], which presumes  
 856 that the distance of two records can be naturally estimated through a certain attribute (e.g., prod-  
 857 uct price). Essentially, it builds a hierarchy of blocks, such that the matching likelihood of two  
 858 descriptions is proportional to the level in which they co-occur for the first time: the blocks at the  
 859 bottom of the hierarchy contain the descriptions with the highest matching likelihood, and vice  
 860 versa for the top hierarchy levels. Then, the hierarchy of blocks is progressively resolved, level by

level, from the leaves to the root. A variation of this approach is presented in [3]: every block is divided into a hierarchy of child blocks and an advanced strategy optimizes their processing on MapReduce.

An entity-centric improvement of the HRP is the *Ordered List of Records* [193], which converts the hierarchy of blocks into a list of records sorted by their likelihood to produce matches. In this way, it trades lower memory consumption for a slightly worse performance than HRP.

Finally, a progressive approach for Multi-source ER over different entity types is proposed in [2]. During the scheduling phase, it divides the total cost budget into several windows of equal cost. For each window, a comparison schedule is generated by choosing the one with the highest expected benefit among those with a cost lower than the current window. The cost of a schedule is computed by considering the cost of finding the description pairs and the cost of resolving them. Its benefit is determined by how many matches are expected to be found by this schedule and how useful they will be to identify more matches within the cost budget. After a schedule is executed, the matching decisions are propagated to all related comparisons so that they are more likely to be chosen by the next schedule. The algorithm terminates upon reaching the cost budget.

## 7.2 Schema-Agnostic Methods

The budget-aware methods for semi-structured data rely on an inherently schema-agnostic functionality that completely disregards any schema information. Thus, they are independent of expert knowledge and require no labeled data for learning how to rank comparisons, blocks, or descriptions.

The cornerstone of sort-based methods is the *Neighbor List* [163], which is created by the schema-agnostic adaptation of Sorted Neighborhood [133]: every token in any attribute value is considered as a blocking key and all descriptions are sorted alphabetically according to these keys. Thus, each description appears in the Neighbor List as many times as the number of its distinct tokens.

The naive progressive approach would be to slide a window of increasing size along this list, incrementally executing the comparisons it defines, as in PSN. This approach, however, results in many repeated comparisons and a random ordering of descriptions with identical keys.

To ameliorate this issue, *Local Schema-agnostic PSN* [163] uses weights based on the assumption that the closer the blocking keys of two descriptions are in the Neighbor List, the more likely they are to be matching. Every comparison defined by the current window size is associated with a numerical estimation of the likelihood that it involves a pair of matches through the schema-agnostic weighting function  $\frac{fr_{i,i}}{fr_i + fr_j - fr_{i,j}}$ , where  $fr_k$  is the number of blocking keys associated with description  $e_k$  (i.e., its occurrences in the Neighbor List), while  $fr_{j,i}$  denotes the frequency of comparison  $\langle e_i, e_j \rangle$  within the current window. All repeated comparisons within every window are eliminated, but there is no way to avoid emitting the same comparison in other window sizes. To address this drawback, *Global Schema-agnostic PSN* [163] defines a global execution order for all comparisons in a specific range of window sizes  $[1, w_{max}]$ , using the same weighting function.

A different approach is implemented by the hash-based method *Progressive Block Scheduling* [163]. First, the input blocks are ordered in increasing cardinality such that the fewer comparisons a block entails, the higher it is ranked. Then, the sorted list of blocks is processed, starting from the top-ranked (i.e., smallest) block. Inside every block, one of Meta-blocking's weighting schemes is used to specify the processing order of comparisons, from the highest weighted to the lowest one. During this process, all repeated comparisons are discarded before computing their weight.

Finally, *Progressive Profile Scheduling* [163] is a hybrid method that relies on the notion of *duplication likelihood*, i.e., the likelihood of an individual description to have one or more matches.



Table 4. A Taxonomy of the Budget-Aware Methods Discussed in Section 7 (in the Order of Presentation)

	Schema-awareness		Key Functionality		Granularity of Functionality			Type of Ordering	
	schema-aware	schema-agnostic	hash-based	sort-based	block-centric	comparison-centric	entity-centric	static	dynamic
Progressive Sorted Neighborhood (PSN) [193]	✓			✓		✓		✓	
Dynamic PSN [143]	✓			✓		✓			✓
Progressive Blocking [143]	✓		✓		✓				✓
Hierarchy of Record Partitions [193]	✓		✓		✓			✓	
Ordered List of Records [193]	✓		✓				✓	✓	
Progressive Relational Entity Resolution [2]	✓		✓			✓			✓
Local Schema-agnostic PSN [163]		✓		✓		✓		✓	
Global Schema-agnostic PSN [163]		✓		✓		✓		✓	
Progressive Block Scheduling [163]		✓	✓		✓	✓		✓	
Progressive Profile Scheduling [163]		✓	✓			✓	✓	✓	

907 This is estimated as the average edge weight of its node in the corresponding blocking graph. This  
 908 method processes the input descriptions in decreasing duplication likelihood. For each description,  
 909 all non-repeated comparisons that entail it are ordered in decreasing weight, as estimated through  
 910 a Meta-blocking weighting scheme, and the top- $k$  ones are emitted.

### 911 7.3 Discussion

912 All budget-aware methods apply ER in a pay-as-you go manner. To address Volume, they all rely  
 913 on blocking methods. The schema-agnostic budget-aware methods are also capable of address-  
 914 ing Variety. Table 4 organizes all methods discussed above into a taxonomy formed by the four  
 915 aforementioned criteria: schema-awareness, functionality of blocking keys, granularity of func-  
 916 tionality, and type of ordering. We observe that there is no dynamic schema-agnostic method that  
 917 adapts its processing order as more duplicates are identified. More research is required toward this  
 918 direction. For dynamic schema-aware methods, a noisy matching method should be used, instead  
 919 of the ideal one that is currently considered. Intelligent ways for tackling the errors introduced by  
 920 noisy matchers are indispensable for a realistic budget-aware scenario.

921 Regarding the relative performance of static methods, the schema-agnostic ones consistently  
 922 outperform the schema-aware ones over several established structured datasets [163]. Among the  
 923 schema-agnostic methods, the two sort-based ones achieve the best performance for structured  
 924 datasets, with the difference between them being statistically insignificant. As a result, Local PSN  
 925 is more suitable in cases of limited memory, but all other settings call for Global PSN, given that it  
 926 avoids multiple emissions of the same comparisons. For large, heterogeneous datasets, Progressive  
 927 Profile Scheduling exhibits the overall best performance, followed by Progressive Block Schedul-  
 928 ing.

## 929 8 INCREMENTAL ER

930 Some Big Data applications need to resolve descriptions that arrive in high Velocity streams or are  
 931 provided as queries against a known entity collection. Rather than a static, offline process over all  
 932 available entity descriptions, such applications process as much entities as needed as long as they  
 933 resolve specific (query) descriptions in (near) real time. The same applies to clean, but evolving  
 934 data repositories, such as data warehouses and knowledge bases, where new entities should be  
 935 incrementally added, without repeating the entire ER process to the already matched descriptions.

936 As an example, consider an application resolving the entities described across news feeds, which  
 937 arrive in a streaming fashion [9, 19, 96]. A journalist using this application could be provided  
 938 with several facts regarding a breaking news story (e.g., persons, buildings, services affected by an  
 939 earthquake), as they get published by different agencies or witnesses, enabling her/him to form a  
 940 complete picture of the events as they occur, in real time. This would require storing only some

parts of the entire entity collection, and discarding the rest, as more descriptions are fed to the system. To evaluate which parts of the collection are more useful to keep, we can design different strategies. For example, we may want to keep the latest entities, since new input entities are more likely to be connected to them. Another strategy would be to keep the entities with many relationships with other entities, since they are more likely to influence the matching decisions.

Such applications call for small memory footprint and low latency, rendering inapplicable the *static* approaches described above. Novel techniques that *dynamically* adapt to data are required. Note that we could distinguish the dynamic methods into those answering to a user-provided query and those resolving streams of entities, but this distinction is orthogonal—streaming methods can be seen as query-based ones that handle streams of queries instead of a single query (e.g., [96]).

### 8.1 Dynamic Blocking

Unlike the works in Section 3, which produce immutable (static) blocks, the dynamic indexing techniques update their blocks, depending on the descriptions that are submitted as queries.

One of the earliest approaches is the *Similarity-aware Index* [36]. The main idea is to pre-calculate similarities between the attribute values that co-occur in blocks in order to avoid similarity calculations at query time, and minimizing response time. This approach uses three indexes that associate blocking keys to attribute values, that contain pre-calculated similarities between attribute values that co-occur in a block, and that associate distinct attribute values with record ids.

This approach is extended by *DySimII* [147] so that all three indexes are updated as query entities arrive. Both its average record insertion time and its average query time remain practically stable, even when the index size grows. Interestingly, the index size can be reduced, without any significant loss in recall, by indexing only a certain portion of the most frequent attribute values.

On another line of research, *F-DySNI* [145, 146] extends the Sorted Neighborhood method by converting the sorted list of blocking keys into an index tree that is faster to search. This is actually a braided AVL tree, i.e., a combination of a height balanced binary tree and a double-linked list [151]: every tree node is linked to its alphabetically sorted predecessor node, to its successor node, and to the list of ids of all entities that correspond to its blocking key. *F-DySNI* actually employs a forest of such index trees, with each tree associated with a different blocking key definition. This forest is updated whenever a query entity arrives and is compatible with both a fixed and an adaptive window. The former defines the rigid number of neighboring nodes that are considered, while the latter considers only the neighbors that exceed a predetermined similarity threshold.

Finally, summarization algorithms for speeding up dynamic ER are presented in [96]. *SkipBloom* summarizes the input descriptions, using their blocking keys, to accelerate comparisons. *BlockSketch* summarizes a block to achieve a fixed number of comparisons per given entity description during Matching, yielding a bounded computational time. Each block is split into sub-blocks based on the distances of the block contents to the blocking key. Each query description is then compared against the sub-block with the smallest distance to its contents. *SBlockSketch* adapts *BlockSketch* to streaming data, maintaining a fixed number of blocks in memory, with a time overhead each time any of those blocks needs to be replaced with blocks residing in secondary storage. To minimize this overhead, a selection algorithm chooses the blocks to be replaced (considering age and size).

### 8.2 Dynamic Matching

These methods resolve online parts of the entity collection that are of interest to a user/application.

*Query-driven ER* [17] uses a two-stage expand-and-resolve query processing strategy. First, it extracts the related descriptions for a query using two expansion operators. Then, it resolves the extracted descriptions collectively, leveraging an existing relevant technique [16]. Due

987 to the complexity of the collective ER strategy, this approach cannot provide real-time answers for  
988 large datasets.

989 In *Query-driven ER with uncertainty* [88], the attribute-level facts for the input entities are associ-  
990 ated with a degree of uncertainty, reflecting the noise from imperfect extraction tools. Matches are  
991 identified using existing ER algorithms and are assigned a probability value. At this offline stage,  
992 no merging takes place. When a query arrives, the descriptions that need to be merged in order to  
993 provide an answer to the query are identified. Then, different merging scenarios are explored and  
994 the one with minimum uncertainty is selected and returned as an answer.

995 *UDD* [168] is an unsupervised method that identifies matches from the results of a query over  
996 multiple Web KBs. First, it removes duplicate descriptions stemming from the same KB, and it  
997 generates a training set. Based on this set of non-matching examples, as well as on similarity  
998 computations between descriptions, it iteratively identifies matches in the query results through  
999 two cooperating classifiers: a weighted component similarity summing and an SVM.

1000 *Sample-and-clean* [182] leverages sampling to improve the quality of aggregate numerical  
1001 queries on large datasets that are too expensive to resolve online. It resolves a small data sample  
1002 and exploits those results to reduce the impact of duplicates on the approximate answers to ag-  
1003 gregate queries.

1004 *QuERy* [5] aims to answer join queries over multiple, overlapping data sources, operating on a  
1005 block level. It identifies which blocks need to be resolved for the requested join and then assumes  
1006 that any matching method can be applied for the matching task.

1007 Complementary to this work, *QDA* [4] tries to reduce the data cleaning overhead and issues  
1008 the minimum number of necessary steps to answer SQL-like selection queries that do not involve  
1009 joins, in an entity-pair level. It performs vestigiality analysis on each block individually to identify  
1010 matching decisions whose answers are guaranteed to not affect the query answers and, thus, need  
1011 not be performed, reducing the matching tasks. In fact, it creates an entity graph for the contents  
1012 of a block and resolves edges belonging to cliques that may affect the query answer. As opposed  
1013 to *Sample-and-Clean* [182], *QDA* provides exact query results.

1014 Finally, *Adaptive Product Normalization* [19] presents an online supervised learning approach  
1015 for resolving different descriptions of the same product. The steps of this approach include (i)  
1016 blocking [118], which defines an initial set of basis functions to compute the similarity between  
1017 specific attributes of the descriptions, (ii) a learning algorithm for training the parameters of a  
1018 composite similarity function, and (iii) clustering [92]. The composite similarity function is trained  
1019 incrementally, using an efficient, online variation of the voted perceptron algorithm [67].

### 1020 8.3 Dynamic Clustering

1021 Special care should be taken to update the entity clusters in an efficient way, as more entities arrive  
1022 in the form of queries or streams. To this end, *Incremental Correlation Clustering* [77] supports all  
1023 kinds of updates (i.e., inserting, deleting, and changing individual descriptions from clusters as well  
1024 as merging and splitting entire clusters), without requiring any prior knowledge of the number of  
1025 clusters. It also allows for fixing prior errors in view of new evidence. Due to its high complexity,  
1026 though, a greedy approximation of polynomial time is also proposed. Constrained versions of  
1027 incremental correlation clustering in other contexts have been proposed in [25, 117].

### 1028 8.4 Discussion

1029 Table 5 organizes all methods discussed in this section into a taxonomy formed by three criteria:  
1030 the ER workflow task corresponding to each method, its schema-awareness, and its algorithmic  
1031 foundation (learning-based or non-learning). These works are crafted for resolving entities in  
1032 (near) real time, not necessarily covering the whole input entity collections, but only a subset

Table 5. A Taxonomy of the Incremental Methods Discussed in Section 8 (in the Order of Presentation)

	Workflow step			Schema awareness		Algorithmic foundation	
	Blocking	Matching	Clustering	Schema-aware	Schema-agnostic	Learning-based	Non-learning
Similarity-Aware Index [36]	✓			✓			✓
DySiml [147]	✓			✓			✓
F-DySNI [145,146]	✓			✓			✓
SBlockSketch [96]	✓	✓		✓			✓
Query-driven Collective ER [17]		✓		✓			✓
Query-driven ER with uncertainty [88]		✓		✓			✓
UDD [168]		✓		✓		Unsupervised	
Sample-and-clean [182]		✓		✓			✓
QuERy [5]		✓		✓			✓
QDA [4]		✓		✓			✓
Adaptive Product Normalization [19]		✓		✓		Supervised	
Incremental Correlation Clustering [77]			✓	✓			✓

that is associated with a user-defined query or a stream of descriptions. In these cases, resolving the whole input set of descriptions would be unnecessarily costly in terms of time and resources. We believe that in the new Big Data era of unprecedented Volume and Velocity, incremental ER methods are becoming far more prevalent, gradually displacing traditional, batch ER methods. Yet, all existing methods are schema-aware, being incapable of addressing Variety. More research is required toward schema-agnostic methods or other approaches that inherently support Variety. This also requires the development of incremental schema-agnostic block processing techniques.

## 9 OTHER ER METHODS

We now cover important ER systems and methods complementary to those presented above.

### 9.1 Deep Learning

The latest developments in deep learning have greatly influenced research in ER. The basic constructs of deep learning methods for ER are Recurrent Neural Networks (RNNs) [59, 196] and word embeddings [13]. RNNs are neural networks with a dynamic temporal behavior. The neurons are fed information not only from the previous layer, but also from their own previous state in time, to process sequences of inputs. Word embeddings are vectorial representations of words, enabling words or phrases to be compared using their vectors. Word embeddings are commonly used with RNNs for speech recognition [121] and similar NLP tasks [32].

*AutoBlock* [202] trains on a set of matches to perform Blocking. First, it converts every token in an attribute value into a word embedding. Then, a neural network combines word embeddings into several attribute embeddings per description, which are fed into multiple indexing functions. The blocking model is learned from training data so that the difference between matching and non-matching descriptions is maximized. LSH is used to detect the most likely matches per description.

*DeepER* [52] explores two methods to generate entity embeddings, i.e., vectorial representations of entity descriptions. The first one exploits word embeddings of tokens appearing in the values of the descriptions, while the latter uses RNNs to convert each description to a vector. *DeepER* can operate both with pre-trained word embeddings [144], and without, proposing ways to create and tune such embeddings, customized for ER. The embedding vector of every description is indexed by LSH, whose parameters are set according to a theoretical analysis and the desired performance.

Q4

1061 Then, each entity creates a block that contains its top-N nearest neighbors. We note that more  
1062 efficient high-dimensional vector similarity methods (than LSH) are now available [53].

1063 *DeepMatcher* [122] extends *DeepER* by introducing an architecture template for deep learning  
1064 ER methods with three main modules: (i) attribute embedding, which converts sequences of words  
1065 used in the attribute values of an entity description to word embedding vectors; (ii) attribute sim-  
1066 ilarity representation, which applies a similarity function on the attribute embeddings of two  
1067 descriptions to obtain a final similarity value of those descriptions (i.e., it learns the similarity  
1068 function); and (iii) a classifier, which uses the similarities between descriptions as features for a  
1069 classifier that decides if a pair of descriptions is a match (i.e., it learns the match function). For  
1070 each module, several options are available. The main ones (e.g., character-level vs. word-level em-  
1071 beddings, pre-trained vs. learned embeddings, fixed vs. learnable similarity function) are used as  
1072 representative points for those modules and are experimentally evaluated, showing their strengths  
1073 and weaknesses.

1074 *Multi-Perspective Matching* [68] adaptively selects (among the similarity measures of *Deep-*  
1075 *Matcher*'s RNN, the Hybrid similarities for textual attributes, and several established approaches  
1076 for string and numeric attributes) the optimal similarity measures for heterogenous attributes.  
1077 First, a unified model for all attributes is built and the supported similarity measures are applied  
1078 to every attribute value pair. A gate mechanism adaptively selects the most appropriate similarity  
1079 measure per attribute and the selected measures are concatenated into a comparison vector. Finally,  
1080 a neural network receives the comparison vector as input and produces the matching probability  
1081 as output.

1082 Other works examine ways of optimizing the use of Deep Learning techniques: to minimize  
1083 the number of required labeled instances, transfer learning is examined in [203] and pre-trained  
1084 subword embeddings are combined with transfer and active learning in [97]; the use of the main  
1085 attention-based transformer architectures is examined in [22]; pre-trained word embeddings are  
1086 coupled with online user reviews for each entity description (e.g., restaurant) in [158].

1087 As we have seen, conventional ER methods identify similar entities based on symbolic features  
1088 (e.g., names, textual descriptions, and attribute values). However, the computation of feature sim-  
1089 ilarity often suffers from the semantic heterogeneity between different Knowledge Graphs (KGs).  
1090 Recently, representation learning techniques have been proposed for Clean-Clean ER, also called  
1091 *Entity Alignment*, where the key idea is to learn embeddings of KGs, such that entities with similar  
1092 neighbor structures in the KG have a close representation in the embedding space. While several  
1093 existing techniques learn entity embeddings in the context of the same KG, doing the same for  
1094 entities of different KGs remains an open challenge. In this setting, *MTransE* [27] learns a map-  
1095 ping between two KG embedding spaces, using a seed set of aligned entities from the two KGs,  
1096 though this is rarely available. *JAPE* [170] jointly trains the attribute and structure embeddings  
1097 using skip-gram and translational models, respectively, to align entities. *GCN-Align* [188] employs  
1098 Graph Convolutional Networks (GCNs) to model entities based on their neighborhood informa-  
1099 tion. However, GCN-Align only considers the equivalent relations between entities, neglecting the  
1100 use of additional KG relationships. *IPTransE* [205] and *BootEA* [171] integrate knowledge among  
1101 different KGs by enlarging the training data (prior alignments) in a bootstrapping way. *KDCoE* [26]  
1102 iteratively co-trains multilingual KG embeddings and fuses them with entity description infor-  
1103 mation for alignment. The above iterative methods improve performance mainly by increasing  
1104 the number of pre-aligned training entity pairs, a strategy that could benefit most alignment ap-  
1105 proaches. Non-iterative methods could achieve better results through bootstrapping.

1106 Methods leveraging additional types of features to refine relation-based embeddings include the  
1107 following. *AttrE* [174] uses character-level literal embeddings over a unified vector space for the  
1108 relationship embeddings after merging the two KGs based on predicate similarity (i.e., predicate

alignment). [201] introduces a framework that unifies multiple views of entities to learn embeddings for entity alignment that is capable of incorporating new features. Specifically, it embeds entities based on the views of entity names, relations, and attributes, with several combination strategies, and considers cross-KG inference methods to enhance the alignment between two KGs. A thorough experimental evaluation of supervised and semi-supervised methods for embedding-based entity alignment has been conducted in [172]. The results on sparse and dense datasets recognize the difficulty of existing methods in aligning (the many) long-tail entities [112]. Finally, we note that the hierarchical structure of KGs (in particular, ontologies) has not been well studied in this context. Thus, more complex KG embeddings (going beyond Euclidean models) are worth exploiting [129].

## 9.2 Crowdsourcing-Based ER Methods

*Crowd-sourcing* is a recent discipline that examines ways of pushing difficult tasks, called *Human Intelligence Tasks* (HITs), to humans, a.k.a., *workers*, at a small price [86]. In the case of ER, one of the most difficult tasks is to decide whether two descriptions match or not. Crowd-sourced ER assumes that humans can improve the effectiveness (i.e., accuracy) of Matching by leveraging contextual information and common sense. Therefore, it asks workers questions about the relation between descriptions for a small compensation per reply. Four main challenges arise in this context:

- Challenge 1: How should HITs be generated? 1126
- Challenge 2: How should HITs be formulated? 1127
- Challenge 3: How can we maximize accuracy, while minimizing the overall monetary cost? 1128
- Challenge 4: How can we restrict the labor cost? 1129

Below, we examine the main solutions to each challenge. 1130

**Challenge 1:** To generate HITs, a hybrid human-machine approach is typically used [28, 113]. First, machine-based techniques are used to do an initial, coarse pass over all pairs of candidate matches, discarding the majority of non-matches, and then, the crowd is asked to verify only the remaining candidate matches. This approach was first introduced by *CrowdER* [181], which automatically computes the similarity between description pairs and discards those below a predetermined threshold. Similarly, *ZenCrowd* [45] combines machine-based pre-processing with crowd-sourced matching, with the latter clarifying low confidence matches produced by the former. A probabilistic framework is used to refine crowd-sourced matches from inconsistent human responses. 1138

**Challenge 2:** Two are the main approaches to formulating HITs [28]: *pair-based* and *cluster-based* (a.k.a. *multi-item*) HITs. The former type asks workers questions of the form “is  $e_i$  matching with  $e_j$ ?” [64, 177, 179, 183, 192], whereas the latter type involves groups with more than two descriptions, requesting workers to mark all duplicates within each group [181]. There is a tradeoff between accuracy and efficiency in terms of cost and time between these two approaches [178]: pair-based HITs are simpler, allowing workers to provide more accurate responses, while the cluster-based HITs enable humans to mark many pairs of records with a few clicks, but their generation constitutes an NP-hard problem that is solved greedily by *CrowdER* [181]. *Hybrid HITs* are used by *Waldo* [178], which argues that the error rate of workers is different for different description pairs. Thus, the high error-rate pairs (i.e., the most “difficult” ones) should be formulated as pair-based HITs, whereas the low error-rate ones should form cluster-based HITs. *Waldo* formalizes the generation of the best hybrid HITs as an optimization task with a specific budget and provides solutions with probabilistic guarantees. Finally, *Crowdlink* [199] decomposes each pair of descriptions into *attribute-level HITs* to facilitate workers when processing descriptions 1152

1153 with overwhelming information, i.e., with complex structures and attributes. A probabilistic  
1154 framework then selects the  $k$  best attributes.

1155 **Challenge 3:** To optimize the tradeoff between accuracy and monetary cost, the transitive rela-  
1156 tion is typically leveraged; if the relation between two descriptions can be inferred by transitivity  
1157 from the already detected duplicates, it is not crowd-sourced. This inference takes two flavors [28]:  
1158 *positive transitivity* suggests that if  $e_i \equiv e_j$  and  $e_j \equiv e_k$ , then  $e_i \equiv e_k$ , whereas *negative transitivity*  
1159 indicates that if  $e_i \equiv e_j$ , but  $e_j \not\equiv e_k$ , then  $e_i \not\equiv e_k$ . These relations lie at the core of several ap-  
1160 proaches [64, 98, 179, 183, 192] that minimize the number of HITs submitted to workers, reducing  
1161 significantly the crowd-sourcing overhead. Their key insight is that finding matches before non-  
1162 matches accelerates the ER process, by making the most of the transitive closure.

1163 Yet, these works assume that workers are infallible, operating as an oracle, which means that un-  
1164 certainty comes exclusively from the machine-generated similarities. In practice, though, the high  
1165 accuracy workers have an error rate up to 25%, due to lack of domain expertise, individual biases,  
1166 tiredness, malicious behaviors, as well as task complexity and ambiguity [185, 197]. When human  
1167 errors occur, the above methods amplify them, thus compromising the overall ER accuracy [185].  
1168 More realistic and robust approaches minimize HITs despite noisy workers, operating on top of a  
1169 *noisy matcher* that introduces uncertainty by returning possibly false results [23, 24, 103, 177, 197].  
1170 Other approaches correct the responses of an oracle through indirect “control queries” [70], or re-  
1171 fine the original crowd-sourced entities based on correlation clustering and additional HITs [185].

1172 **Challenge 4:** A major disadvantage of Crowd-sourced ER is the development cost that is required  
1173 for applying it in practice. To address this issue, *Corleone* [74] implements a hands-off crowd-  
1174 sourcing solution for the entire ER workflow that involves no software developers. It automatically  
1175 generates blocking rules, learns a matcher from the HITs that are iteratively answered by work-  
1176 ers (active learning minimizes the monetary cost), and finally returns the equivalence clusters.  
1177 However, *Corleone* does not scale to large datasets, as it exclusively runs in-memory on a sin-  
1178 gle machine. To address this issue, *Falcon* [42] runs *Corleone* on a MapReduce cluster, exploiting  
1179 crowd-time to run machine tasks. Experiments have shown that it scales to 2.5 million descriptions  
1180 in 2–14 hours for only ~\$60. *CloudMatcher* [76] goes one step further, implementing *Falcon* as a  
1181 cloud service.

### 1182 9.3 Rule-Based ER Methods

1183 This category includes methods that leverage the knowledge of domain experts, who can provide  
1184 some generic initial rules (e.g., “if two descriptions have a similar address value, then they are  
1185 matches”) that will help an ER algorithm to find some or all matches in a given task.

1186 *HIL* [83] is a high-level scripting language for expressing such rules. A HIL program determines  
1187 complex ER pipelines, capturing the overall integration flow through a combination of SQL-like  
1188 rules that link, map, fuse, and aggregate descriptions. Its data model makes uses of logical indices  
1189 to facilitate the modular construction and aggregation of complex entity descriptions. Its flexible,  
1190 open type system allows HIL to handle irregular, sparse, or partially known input data.

1191 Reasoning and discovery techniques have also been proposed for automatically obtaining more  
1192 matching rules. Dependency-based reasoning techniques to help define keys for Matching and  
1193 Blocking are introduced in [61, 62]. At their core lie *matching dependencies* (MDs), which allow  
1194 one to infer matches, based on the similarity of database records on some attributes in relational  
1195 schemata. MDs can be used in both Blocking and Matching to directly infer matches, but they  
1196 can also be extended and used to infer new MDs, minimizing manual effort and leading to more  
1197 matches.

Even though the MDs are looser versions of the strict functional dependencies in relational databases, they may still be too strict in practice. To address this issue, the *conditional MDs* (CMDs) [187] bind MDs to a certain subset of descriptions in a relational table and have more expressive power than MDs for declaring constraints with conditions, allowing a wider range of applications.

*Certus* [110] introduces *graph differential dependencies* (GDDs) as an extension of MDs and CMDs that enables approximate matching of values. It adopts a graph model for entity descriptions, which enables the formal representation of descriptions even in unstructured sources, while a specialized algorithm generates a non-redundant set of GDDs from labeled data. *Certus* employs the learned GDDs for improving the accuracy of ER results. Unlike MDs and CMDs, which operate only on structured data, *Certus* can identify matches irrespective of structure and with no assumed schema.

#### 9.4 Temporal ER Methods

Entity descriptions are often associated with temporal information in the form of timestamps (e.g., user log data or sensor data) [31, 123] or temporal validity of attributes (e.g., population, marital status, affiliation) [85]. ER methods exploiting such temporal information may show better performance than those ignoring it [30]; rather than deciding if two descriptions match, they try to decide if a new description matches with a set of descriptions that have been already identified as matches. The probability of a value re-appearing over time is examined in [30]. Intuitively, a description might change its attribute values in a way that is dependent on previous values. For example, if a person’s location has taken the values Los Angeles, San Francisco, San Jose in the past, then these values are more likely to appear in this person’s future location than Berlin or Cairo. *SFDS* [31] follows a “static first, dynamic second” strategy: initially, it assumes that all descriptions are static (i.e., not evolving over time) and groups them into clusters. These are later merged in the dynamic phase, if the different clusters correspond to the same entities that have evolved over time.

#### 9.5 Open-Source ER Tools

We now elaborate on the main systems that are crafted for end-to-end Entity Resolution. We examined the 18 non-commercial and 15 commercial tools that are listed in the extended version of [104]<sup>8</sup> along with the 10 Link Discovery frameworks surveyed in [127]. Among them, we exclusively consider the open-source systems, since the closed-code and the commercial ones provide insufficient information about their internal functionality and/or their algorithms.

A summary of the main open-source ER systems appears in Table 6. For each one, we report whether it involves one or more methods per workflow step of the general end-to-end ER pipeline in Figure 2(a), whether it supports parallelization, budget-aware or incremental methods, a graphical user interface (GUI), as well as its programming language. To facilitate their understanding, we group all systems into three categories, depending on their input data: (i) systems for structured data, (ii) systems for semi-structured data, and (iii) hybrid systems.

The tools for structured data include *Dedupe* [20], *FRIL* [93], *OYSTER* [125], *RecordLinkage* [156], *DuDe* [51], *Febrl* [33], *Magellan* [104], and *FAMER* [153]. All of them offer at least one method for Blocking and Matching, while disregarding Clustering. The only exception is *FAMER*, which exclusively focuses on Clustering, implementing several established techniques in Apache Flink. *Febrl* involves the richest variety of non-learning, schema-aware Blocking methods, which can be combined with several similarity measures and top-performing classifiers for supervised

<sup>8</sup><http://pages.cs.wisc.edu/~anhai/papers/magellan-tr.pdf>.



Table 6. The Main Open-Source ER Tools (a Feature in Parentheses is Partially Supported)

Tool	Blocking	Block Processing	Matching	Clustering	Parallelization	Budget-aware ER	Incremental ER	GUI	Language
Dedupe [20]	✓	-	✓	-	multi-core	-	-	-	Python
DuDe [51]	✓	-	✓	-	-	-	-	-	Java
Febrl [33]	✓	-	✓	-	multi-core	-	-	✓	Python
FRIL [93]	✓	-	✓	-	-	-	-	✓	Java
OYSTER [125]	✓	-	✓	-	-	-	-	-	Java
RecordLinkage [156]	✓	-	✓	-	-	-	-	-	R
Magellan [104]	✓	-	✓	-	(Apache Spark)	-	-	✓	Python
FAMER [153]	-	-	-	✓	Apache Flink	-	-	-	Java
Silk [91]	✓	-	✓	-	Apache Spark	-	-	✓	Scala
LIMES [128]	✓	-	✓	-	(multi-core)	-	-	✓	Java
Duke	✓	-	✓	-	-	-	✓	-	Java
KnoFuss [130]	✓	-	✓	-	-	-	-	-	Java
SERIMI [8]	✓	-	✓	-	-	-	-	-	Ruby
MinoanER [56]	✓	✓	✓	-	Apache Spark	-	-	-	Java
JedAI [142]	✓	✓	✓	✓	Apache Spark	✓	-	✓	Java

1241 matching. Magellan conveys a Deep Learning module, which is a unique feature among all ER  
 1242 tools. Most systems are implemented in Java or Python, with just three of them offering a GUI.

1243 The systems for semi-structured data receive as input RDF dump files or SPARQL endpoints.  
 1244 The most prominent ones are Silk [91] and LIMES [128], which are crafted for the Link Discovery  
 1245 problem (i.e., the generic task of identifying relations between entities). Restricting them to the  
 1246 discovery of sameAs relations renders them suitable for ER. Both systems involve custom blocking  
 1247 techniques along with a large variety of character- and token-based similarity measures. Combi-  
 1248 nations of these similarity measures are learned in a (semi-)supervised way for effective Matching.  
 1249 Both tools offer an intuitive GUI, unlike the remaining ones, namely, SERIMI [8], Duke,<sup>9</sup> and Kno-  
 1250 Fuss [130]. These systems merely apply simple Blocking techniques to literal values and focus  
 1251 primarily on Matching, providing effective, but custom techniques based on similarity measures.

1252 The hybrid tools, MinoanER [56] and JedAI [142], apply uniformly to both structured and semi-  
 1253 structured data. This is possible due to the schema-agnostic functionality of their methods. In fact,  
 1254 they implement the main non-learning, schema-agnostic techniques for Blocking, Matching, and  
 1255 Clustering. They are also the only systems that offer Block Processing techniques.

1256 Overall, we observe that all open-source systems focus on Matching, conveying a series of string  
 1257 similarity measures for the comparison of attribute values. More effort should be spent on cover-  
 1258 ing more adequately all workflow steps of the general end-to-end ER workflow. Most importantly,  
 1259 except for Duke's Incremental ER and JedAI's Progressive ER, no system supports any other pro-  
 1260 cessing mode other than budget-agnostic ER. This should be addressed in the future.

## 1261 9.6 Discussion

1262 Even though Rule-based and Temporal ER constitute important topics, more effort is lately directed  
 1263 at leveraging Deep Learning techniques for ER. These efforts have already paid off, as the result-  
 1264 ing techniques achieve the state-of-the-art performance for several established benchmark datasets  
 1265 [122], outperforming methods based on traditional machine learning. Yet, the time efficiency and  
 1266 the availability of a representative set of labeled instances remain important issues. The latter is

<sup>9</sup><https://github.com/larsga/Duke>.

intelligently addressed by a series of Crowdsourcing-based ER methods. Despite the considerable recent advancements, though, Crowdsourced ER still suffers from significant monetary cost and high latency, while it can only be used by expert users. Systems like CloudMatcher contribute to its democratization, while systems like MinoanER and JedAI aim to act as libraries of the state-of-the-art methods for end-to-end ER over Big Data.

## 10 DIRECTIONS FOR FUTURE WORK

As we have just begun to realize the need for *Entity Resolution Management Systems* [104], we next highlight a few critical research directions for future work, which aim to support advanced services for specifying, maintaining, and making accountable complex ER workflows.

**Multi-modal ER.** In the Big Data era, multi-modal entity descriptions are becoming increasingly common. Factual, textual, or image-based descriptions of the same real-world entities are available from different sources and at different temporal or spatial resolutions. Each modality carries a specific piece of information about an entity and offers added value that cannot be obtained from the other modalities. Recent years have witnessed a surge of the need to jointly analyze multi-modal descriptions [204]. Finding semantically similar descriptions from different modalities is one of the core problems of multi-modal learning. Most current approaches focus on how to utilize extrinsic supervised information to project one modality to the other, or map two modalities into a commonly shared space. The performance of these methods heavily depends on the richness of the training samples. In real-world applications though, obtaining matched data from multiple modalities is costly, or impossible [71]. Thus, we need sample-insensitive methods for multi-modal ER, and in this respect, we can leverage recent advances in multi-modal ML techniques [11].

**Debugging and Repairing ER Workflows.** Current ER research mainly focuses on developing accurate and efficient techniques, which in reality are constrained by a number of factors, such as low quality entity descriptions, ambiguous domain knowledge, and limited ground truth. Hence, it is difficult to guarantee the quality of ER workflows at specification time. To support a *continuous* specification of ER workflows, an iterative approach is needed to refine ER workflows by identifying and analyzing the mistakes (false matches and non-matches) of ER enactments at each iteration step. Debugging ER workflows requires one to (a) understand the mistakes made by Blocking or Matching algorithms; (b) diagnose root causes of these mistakes (e.g., due to dirty data, problematic feature sets, or even tuning parameters of algorithms); and (c) prioritize mistakes and take actions to fix them [104]. We note that not all categories of mistakes have the same impact on the end-to-end quality of ER workflows. For example, the removal of outliers from input data often leads to overfitting problems of learning-based matchers. Recognizing patterns of mistakes reproduced under similar conditions can provide valuable insights in order to repair ER workflows. The focus of ER work so far was in preventing rather than repairing mistakes in ER results. Recent work on debugging and repairing Big Data analytics pipelines can be leveraged in this respect [39, 78, 115].

**Fairness in Long Tail Entities Resolution.** The reported accuracy scores of several ER approaches are fairly high, giving the impression that the problem is well-understood and solved. At the same time, recent works (e.g., [60, 176]) claim that ER systems base their performance on entity popularity, while their performance drops significantly when focusing on the rare, long tail entities. However, the lack of formal definitions regarding what is popular and long tail entities for the ER task prevents the identification of the difficult cases for ER, for which systems need to be adapted or new approaches need to be developed [186]. Better understanding of such cases will be helpful for ER, since knowledge about long tail entities is less accessible, not redundant, and hard to obtain.

1313 **Diversity of Matching Entities.** Works in budget-aware ER typically focus on maximizing the  
1314 reported matches, by potentially exploiting the partial matching results obtained so far in an itera-  
1315 tive process. Then, it will be interesting to measure the added knowledge that the ER process could  
1316 achieve after merging the matches, similar to the notion of diversity in information retrieval. Our  
1317 intuition is that merges resulting from somehow similar entities are more beneficial when com-  
1318 pared to merges from strongly similar entities. Thus, given a constraint in the number of possible  
1319 merges, the goal is to perform those that contribute most in diversifying the knowledge encoded  
1320 in the result. Added knowledge can be measured by the number of relationships of a merged en-  
1321 tity with other entities. We consider such relationships as a unit of knowledge increase: when two  
1322 relationships represent two different knowledge units, they are both useful; when they overlap,  
1323 they represent the same knowledge unit, so we do not gain by knowing both of them.

1324 **Bias in ER.** Similarity measures lie at the core of Matching. However, it is well known that not  
1325 all similarity measures are appropriate for all types of data (e.g., strings, locations, and videos).  
1326 Moreover, when focusing on particular types of measures, e.g., measures for string matching, we  
1327 do not know beforehand which is the ideal measure for counting similarities with respect to the  
1328 semantics of the strings to be compared. For instance, we possibly need different measures for  
1329 computing similarities between American names than for Chinese names. In such scenarios, we  
1330 typically exploit some solid empirical evidence, which, based on some of the characteristics that  
1331 our data have, leads us to select, unintentionally, a particular measure. This fact can be considered  
1332 as algorithmic bias [79]. As a first step, for achieving unbiased and fair results, it is important to  
1333 experimentally study if there is bias in ER algorithms [7, 95]. Moving forward to the next genera-  
1334 tion of approaches, we need to propose solutions and provide guidelines that make ER algorithms  
1335 fair.

## 1336 11 CONCLUSIONS

1337 Although ER has been studied for more than three decades in different computer science commu-  
1338 nities, it still remains an active area of research. The problem has enjoyed a renaissance during  
1339 recent years, with the avalanche of data-intensive descriptions of real-world entities provided by  
1340 government, scientific, corporate, or even user-crafted data sources. Reconciling different entity  
1341 descriptions in the Big Data era poses new challenges both at the algorithmic and the system level:  
1342 Volume, due to the very high number of entities and data sources; Variety, due to the extreme  
1343 schema heterogeneity; Velocity, due to the continuously increasing volume of data; and Veracity,  
1344 due to the high level of noise and inconsistencies. In this survey, we have focused on how the main  
1345 algorithms in each step of the end-to-end ER workflow address the combination of these chal-  
1346 lenges. Blocking and Block Processing, two steps that by definition tackle Volume, also address  
1347 Variety mainly through a schema-agnostic, non-learning functionality. Most Matching methods  
1348 employ a schema-agnostic, collective functionality, which leverages information provided by re-  
1349 lated entities, in order to address Variety and Veracity. Budget-aware ER methods rely on Block-  
1350 ing and a usually schema-agnostic functionality to simultaneously address Volume and Variety,  
1351 while Incremental Methods address Volume and Velocity through Blocking, but their schema-  
1352 aware functionality prevents them from tackling Variety, too. In all cases, massive parallelization,  
1353 usually through the MapReduce framework, plays an important role in further improving scala-  
1354 bility and, thus, addressing Volume. Note, though, that we share the view of ER as an engineering  
1355 task by nature, and hence, we cannot just keep developing ER algorithms in a vacuum [104]. In  
1356 the Big Data era, we opt for *open-world ER systems* that allow one to plug-and-play different algo-  
1357 rithms and that can easily integrate with third-party tools for data exploration, data cleaning, or  
1358 data analytics.

## REFERENCES

- [1] Akiko N. Aizawa and Keizo Oyama. 2005. A fast linkage detection scheme for multi-source information integration. In *WIRI*. 30–39. 1359  
1360
- [2] Yasser Altowim, Dmitri V. Kalashnikov, and Sharad Mehrotra. 2014. Progressive approach to relational entity resolution. *PVLDB* 7, 11 (2014), 999–1010. 1361  
1362
- [3] Yasser Altowim and Sharad Mehrotra. 2017. Parallel progressive approach to entity resolution using MapReduce. In *ICDE*. 909–920. 1363  
1364
- [4] Hotham Altwaijry, Dmitri V. Kalashnikov, and Sharad Mehrotra. 2017. QDA: A query-driven approach to entity resolution. *TKDE* 29, 2 (2017), 402–417. 1365  
1366
- [5] Hotham Altwaijry, Sharad Mehrotra, and Dmitri V. Kalashnikov. 2015. QuERY: A framework for integrating entity resolution with query processing. *PVLDB* 9, 3 (2015), 120–131. 1367  
1368
- [6] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. 2002. Eliminating fuzzy duplicates in data warehouses. In *VLDB*. 586–597. 1369  
1370
- [7] Rico Angell, Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2018. Themis: Automatically testing software for discrimination. In *the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 871–875. 1371  
1372
- [8] Samur Araújo, Duc Thanh Tran, Arjen P. de Vries, and Daniel Schwabe. 2015. SERIMI: Class-based matching for instance matching across heterogeneous datasets. *TKDE* 27, 5 (2015), 1397–1410. 1373  
1375
- [9] Tiago Brasileiro Araújo, Kostas Stefanidis, Carlos Eduardo Santos Pires, Jyrki Nummenmaa, and Thiago Pereira da Nóbrega. 2020. Schema-agnostic blocking for streaming data. In *Proceedings of the 35th ACM/SIGAPP Symposium on Applied Computing (SAC’20)*. 412–419. 1376  
1377
- [10] Javed A. Aslam, Ekaterina Pelekhev, and Daniela Rus. 2004. The star clustering algorithm for static and dynamic information organization. *J. Graph Algorithms Appl.* 8 (2004), 95–129. 1378  
1380
- [11] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. 2019. Challenges and applications in multimodal machine learning. In *The Handbook of Multimodal-Multisensor Interfaces*. ACM and Morgan & Claypool, 17–48. 1381  
1382
- [12] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. 2004. Correlation clustering. *Machine Learning* 56, 1–3 (2004), 89–113. 1383  
1384
- [13] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *JMLR* 3 (2003), 1137–1155. 1385  
1386
- [14] Omar Benjelloun, Hector Garcia-Molina, Heng Gong, Hideki Kawai, Tait Elliott Larson, David Menestrina, and Sutthipong Thavisomboon. 2007. D-Swoosh: A family of algorithms for generic, distributed entity resolution. In *IEEE Conference on Distributed Computing Systems (ICDCS’07)*. 37. 1387  
1388
- [15] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. 2009. Swoosh: A generic approach to entity resolution. *VLDB J.* 18, 1 (2009), 255–276. 1389  
1390
- [16] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *TKDD* 1, 1 (2007), 5. 1391  
1392
- [17] Indrajit Bhattacharya and Lise Getoor. 2007. Query-time entity resolution. *J. Artif. Intell. Res.* 30 (2007), 621–657. 1393
- [18] Guilherme Dal Bianco, Marcos André Gonçalves, and Denio Duarte. 2018. BLOSS: Effective meta-blocking with almost no effort. *Inf. Syst.* 75 (2018), 75–89. 1394  
1395
- [19] Mikhail Bilenko, Sugato Basu, and Mehran Sahami. 2005. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM’05)*. 58–65. 1396  
1397
- [20] M. Bilenko and R. J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*. 1398  
1400
- [21] Christoph Böhm, Gerard de Melo, Felix Naumann, and Gerhard Weikum. 2012. LINDA: Distributed web-of-data-scale entity matching. In *CIKM*. 1401  
1402
- [22] Ursin Brunner and Kurt Stockinger. 2020. Entity matching with transformer architectures—A step forward in data integration. In *Proceedings of the 23rd International Conference on Extending Database Technology (EDBT’20)*. 463–473. 1403  
1404
- [23] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2016. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD*. 1405  
1407
- [24] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2018. A partial-order-based framework for cost-effective crowdsourced entity resolution. *VLDB J.* 27, 6 (2018), 745–770. 1408  
1409
- [25] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. 2004. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.* 33, 6 (2004), 1417–1440. 1410  
1411
- [26] Muhao Chen, Yingtao Tian, Kai-Wei Chang, Steven Skiena, and Carlo Zaniolo. 2018. Co-training embeddings of knowledge graphs and entity descriptions for cross-lingual entity alignment. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI’18)*. 1412  
1413  
1414

- 1415 [27] Muhao Chen, Yingtao Tian, Mohan Yang, and Carlo Zaniolo. 2017. Multilingual knowledge graph embeddings for  
 1416 cross-lingual knowledge alignment. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*  
 1417 (*IJCAI'17*).
- 1418 [28] Xiao Chen. 2015. Crowdsourcing entity resolution: A short overview and open issues. In *GvDB*. 72–77.
- 1419 [29] Xiao Chen, Eike Schallehn, and Gunter Saake. 2018. Cloud-scale entity resolution: Current state and open challenges.  
 1420 *OJBD* 4, 1 (2018).
- 1421 [30] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F. Naughton. 2014. Modeling entity evolution for temporal record  
 1422 matching. In *SIGMOD*. 1175–1186.
- 1423 [31] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F. Naughton. 2014. Tracking entities in the dynamic world: A fast  
 1424 algorithm for matching temporal records. *PVLDB* 7, 6 (2014), 469–480.
- 1425 [32] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and  
 1426 Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine transla-  
 1427 tion. In *EMNLP*. 1724–1734.
- 1428 [33] P. Christen. 2008. Febrl—An open source data cleaning, deduplication and record linkage system with a graphical  
 1429 user interface. In *Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data Mining*  
 1430 (*KDD'08*). 1065–1068.
- 1431 [34] Peter Christen. 2012. *Data Matching*. Springer.
- 1432 [35] Peter Christen. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE* 24, 9  
 1433 (2012), 1537–1555.
- 1434 [36] Peter Christen, Ross W. Gayler, and David Hawking. 2009. Similarity-aware indexing for real-time entity resolution.  
 1435 In *CIKM*. 1565–1568.
- 1436 [37] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. 2015. *Entity Resolution in the Web of Data*. Morgan  
 1437 & Claypool.
- 1438 [38] Xu Chu, Ihab F. Ilyas, and Paraschos Koutris. 2016. Distributed data deduplication. *PVLDB* 9, 11 (2016), 864–875.
- 1439 [39] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, K. Tae, and Steven Euijong Whang. 2019. Slice finder: Automated  
 1440 data slicing for model validation. In *ICDE*.
- 1441 [40] Aaron Clauset, Mark E. J. Newman, and Christopher Moore. 2004. Finding community structure in very large net-  
 1442 works. *Physical Review E* 70, 6 (2004), 066111.
- 1443 [41] William W. Cohen and Jacob Richman. 2002. Learning to match and cluster large high-dimensional data sets for  
 1444 data integration. In *SIGKDD*.
- 1445 [42] Sanjib Das, Paul Suganthan G. C., AnHai Doan, Jeffrey F. Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute,  
 1446 Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling up hands-off crowdsourced entity matching to build  
 1447 cloud services. In *SIGMOD*. 1431–1446.
- 1448 [43] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Commun. ACM*  
 1449 51, 1 (2008), 107–113.
- 1450 [44] Jeremy Debattista, Christoph Lange, Sören Auer, and Dominic Cortis. 2018. Evaluating the quality of the LOD cloud:  
 1451 An empirical investigation. *Semantic Web* 9, 6 (2018), 859–901.
- 1452 [45] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2013. Large-scale linked data integration  
 1453 using probabilistic reasoning and crowdsourcing. *Vldb J.* 22, 5 (2013), 665–687.
- 1454 [46] Juan A. Díaz and Elena Fernández. 2001. A Tabu search heuristic for the generalized assignment problem. *EJOR* 132,  
 1455 1 (2001), 22–38.
- 1456 [47] Dimas Cassimiro do Nascimento, Carlos Eduardo Santos Pires, and Demetrio Gomes Mestre. 2020. Exploiting block  
 1457 co-occurrence to control block sizes for entity resolution. *Knowl. Inf. Syst.* 62, 1 (2020), 359–400.
- 1458 [48] Xin Dong, Alon Y. Halevy, and Jayant Madhavan. 2005. Reference reconciliation in complex information spaces. In  
 1459 *SIGMOD*. 85–96.
- 1460 [49] Xin Luna Dong and Divesh Srivastava. 2015. *Big Data Integration*. Morgan & Claypool.
- 1461 [50] Carina Friedrich Dorneles, Rodrigo Gonçalves, and Ronaldo dos Santos Mello. 2011. Approximate data instance  
 1462 matching: A survey. *KAIS* 27, 1 (01 Apr. 2011), 1–21.
- 1463 [51] Uwe Draisbach and Felix Naumann. 2010. DuDe: The duplicate detection toolkit. In *QDB*.
- 1464 [52] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Dis-  
 1465 tributed representations of tuples for entity resolution. *PVLDB* 11, 11 (2018), 1454–1467.
- 1466 [53] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2019. Return of the Lernaean  
 1467 Hydra: Experimental evaluation of data series approximate similarity search. *PVLDB* 13, 3 (2019).
- 1468 [54] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. 2017. Matching web  
 1469 tables with knowledge base entities: From entity lookups to entity embeddings. In *ISWC*. 260–277.
- 1470 [55] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. 2017. Parallel  
 1471 meta-blocking for scaling entity resolution over big heterogeneous data. *Inf. Syst.* 65 (2017), 137–157.

- [56] Vasilis Efthymiou, George Papadakis, Kostas Stefanidis, and Vassilis Christophides. 2019. MinoanER: Schema-agnostic, non-iterative, massively parallel resolution of web entities. In *EDBT*. 373–384. 1472–1473
- [57] Vasilis Efthymiou, Kostas Stefanidis, and Vassilis Christophides. 2015. Big data entity resolution: From highly to somehow similar entity descriptions in the Web. In *IEEE Big Data*. 401–410. 1474–1475
- [58] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate record detection: A survey. *TKDE* 19, 1 (2007), 1–16. 1476–1477
- [59] Jeffrey L. Elman. 1990. Finding structure in time. *Cogn. Sci.* 14, 2 (1990), 179–211. 1478
- [60] José Esquivel, Dyaa Albakour, Miguel Martinez-Alvarez, David Corney, and Samir Moussa. 2017. On the long-tail entities in news. In *ECIR*. 1479–1480
- [61] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520. 1481–1482
- [62] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. 2009. Reasoning about record matching rules. *PVLDB* 2, 1 (2009), 407–418. 1483–1484
- [63] I. P. Fellegi and A. B. Sunter. 1969. A theory for record linkage. *J. Amer. Statist. Assoc.* 64 (1969), 1183–1210. 1485
- [64] Donatella Firmani, Barna Saha, and Divesh Srivastava. 2016. Online entity resolution using an oracle. *PVLDB* 9, 5 (2016), 384–395. 1486–1487
- [65] Jeffrey Fisher, Peter Christen, Qing Wang, and Erhard Rahm. 2015. A clustering-based framework to control block sizes for entity resolution. In *Proceedings of the 21th ACM International Conference on Knowledge Discovery and Data Mining (KDD’15)*. 279–288. 1488–1489
- [66] Gary William Flake, Robert Endre Tarjan, and Kostas Tsioutsoulis. 2003. Graph clustering and minimum cut trees. *Internet Mathematics* 1, 4 (2003), 385–408. 1491–1492
- [67] Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning* 37, 3 (1999), 277–296. 1493–1494
- [68] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. End-to-end multi-perspective matching for entity resolution. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI’19)*. 4961–4967. 1495–1496
- [69] Avigdor Gal. 2014. Tutorial: Uncertain entity resolution. *PVLDB* 7, 13 (2014), 1711–1712. 1497–1498
- [70] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. 2018. Robust entity resolution using random graphs. In *SIGMOD*. 3–18. 1499–1500
- [71] Nengneng Gao, Sheng-Jun Huang, Yifan Yan, and Songcan Chen. 2018. Cross modal similarity learning with active queries. *Pattern Recogn.* 75, C (2018), 214–222. 1501–1502
- [72] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: Theory, practice & open challenges. *PVLDB* 5, 12 (2012), 2018–2019. 1503–1504
- [73] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. 2007. Clustering aggregation. *TKDD* 1, 1 (2007), 4. 1505
- [74] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD*. 601–612. 1506–1507
- [75] Behzad Golshan, Alon Y. Halevy, George A. Mihaila, and Wang-Chiew Tan. 2017. Data integration: After the teenage years. In *PODS*. 101–106. 1508–1509
- [76] Yash Govind, Erik Paulson, Palaniappan Nagarajan, Paul Suganthan G. C., AnHai Doan, Youngchoon Park, Glenn Fung, Devin Conathan, Marshall Carter, and Mingju Sun. 2018. CloudMatcher: A hands-off cloud/crowd service for entity matching. *PVLDB* 11, 12 (2018), 2042–2045. DOI : <https://doi.org/10.14778/3229863.3236255> 1510–1512
- [77] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. 2014. Incremental record linkage. *PVLDB* 7, 9 (May 2014), 697–708. 1513–1514
- [78] M. A. Gulzar, M. Interlandi, S. Yoo, S. D. Tetali, T. Condie, T. Millstein, and M. Kim. 2016. BigDebug: Debugging primitives for interactive big data processing in Spark. In *Proceedings of the 38th International Conference on Software Engineering (ICSE’16)*. 784–795. 1515–1517
- [79] Sara Hajian, Francesco Bonchi, and Carlos Castillo. 2016. Algorithmic bias: From discrimination discovery to fairness-aware data mining. In *KDD*. 1518–1519
- [80] Otkie Hassanzadeh, Fei Chiang, Renée J. Miller, and Hyun Chul Lee. 2009. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB* 2, 1 (2009), 1282–1293. 1520–1521
- [81] Otkie Hassanzadeh and Renée J. Miller. 2009. Creating probabilistic databases from duplicated data. *VLDB J.* 18, 5 (2009), 1141–1166. 1522–1523
- [82] Taher H. Haveliwala, Aristides Gionis, and Piotr Indyk. 2000. Scalable techniques for clustering the Web. In *WebDB*. 129–134. 1524–1525
- [83] Mauricio A. Hernández, Georgia Koutrika, Rajasekar Krishnamurthy, Lucian Popa, and Ryan Wisnesky. 2013. HIL: A high-level scripting language for entity integration. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT’13)*. 549–560. 1526–1527

- 1529 [84] Mauricio A. Hernández and Salvatore J. Stolfo. 1995. The merge/purge problem for large databases. In *SIGMOD*.  
1530 127–138.
- 1531 [85] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and tem-  
1532 porally enhanced knowledge base from Wikipedia. *Artif. Intell.* 194 (2013), 28–61.
- 1533 [86] Jeff Howe. 2006. The rise of crowdsourcing. *Wired Magazine* 14, 6 (2006), 1–4.
- 1534 [87] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning*. ACM.
- 1535 [88] Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, and Yannis Velegarakis. 2010. On-the-fly entity-aware query  
1536 processing in the presence of linkage. *PVLDB* 3, 1 (2010), 429–438.
- 1537 [89] Ekaterini Ioannou, Claudia Niederée, and Wolfgang Nejdl. 2008. Probabilistic entity linkage for heterogeneous in-  
1538 formation spaces. In *CAiSE*.
- 1539 [90] Ekaterini Ioannou, Nataliya Rassadko, and Yannis Velegarakis. 2013. On generating benchmark data for entity match-  
1540 ing. *J. Data Semantics* 2, 1 (2013), 37–56.
- 1541 [91] Robert Isele and Christian Bizer. 2012. Learning expressive linkage rules using genetic programming. *PVLDB* 5, 11  
1542 (2012), 1638–1649.
- 1543 [92] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. 1999. Data clustering: A review. *ACM Comput. Surv.* 31, 3  
1544 (1999), 264–323.
- 1545 [93] Pawel Jurczyk, James J. Lu, Li Xiong, Janet D. Cragan, and Adolfo Correa. 2008. Fine-grained record integration and  
1546 linkage tool. *BDR* 82, 11 (2008).
- 1547 [94] Anna Jurek, Jun Hong, Yuan Chi, and Weiru Liu. 2017. A novel ensemble learning approach to unsupervised record  
1548 linkage. *Inf. Syst.* 71 (2017), 40–54.
- 1549 [95] Alexandros Karakasidis and Evaggelia Pitoura. 2019. Identifying bias in name matching tasks. In *EDBT*. 626–629.
- 1550 [96] Dimitrios Karapiperis, Aris Gkoulalas-Divanis, and Vassilios S. Verykios. 2018. Summarization algorithms for record  
1551 linkage. In *EDBT*. 73–84.
- 1552 [97] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource deep entity resolution  
1553 with transfer and active learning. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*  
1554 (*ACL '19*). 5851–5861.
- 1555 [98] Xiangyu Ke, Michelle Teo, Arijit Khan, and Vijaya Krishna Yalavarthi. 2018. A demonstration of PERC: Probabilistic  
1556 entity resolution with crowd errors. *PVLDB* 11, 12 (2018), 1922–1925.
- 1557 [99] Mayank Kejriwal and Daniel P. Miranker. 2013. An unsupervised algorithm for learning blocking schemes. In *ICDM*.  
1558 340–349.
- 1559 [100] Mayank Kejriwal and Daniel P. Miranker. 2014. A two-step blocking scheme learner for scalable link discovery. In  
1560 *OM*. 49–60.
- 1561 [101] Mayank Kejriwal and Daniel P. Miranker. 2015. A DNF blocking scheme learner for heterogeneous datasets. *CoRR*  
1562 abs/1501.01694.
- 1563 [102] Mayank Kejriwal and Daniel P. Miranker. 2015. An unsupervised instance matcher for schema-free RDF data. *J. Web*  
1564 *Sem.* 35 (2015), 102–123.
- 1565 [103] Asif R. Khan and Hector Garcia-Molina. 2016. Attribute-based crowd entity resolution. In *CIKM*. 549–558.
- 1566 [104] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalani, Jeffrey R. Ballard, Han Li, Fatemah  
1567 Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra.  
1568 2016. Magellan: Toward building entity matching management systems. *PVLDB* 9, 12 (2016).
- 1569 [105] Hanna Köpcke and Erhard Rahm. 2010. Frameworks for entity matching: A comparison. *Data Knowl. Eng.* 69, 2  
1570 (2010), 197–210.
- 1571 [106] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world  
1572 match problems. *PVLDB* 3, 1 (2010).
- 1573 [107] Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. 2006. Record linkage: Similarity measures and algorithms. In  
1574 *SIGMOD*. 802–803.
- 1575 [108] Jerome M. Kurtzberg. 1962. On approximation methods for the assignment problem. *J. ACM* 9, 4 (1962), 419–439.
- 1576 [109] Shrinu Kushagra, Hemant Saxena, Ihab F. Ilyas, and Shai Ben-David. 2019. A semi-supervised framework of clus-  
1577 tering selection for de-duplication. In *Proceedings of the 35th IEEE International Conference on Data Engineering*  
1578 (*ICDE'19*). 208–219.
- 1579 [110] Selasi Kwashie, Jixue Liu, Jiuyong Li, Lin Liu, Markus Stumptner, and Lujing Yang. 2019. Certus: An effective entity  
1580 resolution approach with graph differential dependencies (GDDs). *PVLDB* 12, 6 (2019), 653–666.
- 1581 [111] Simon Lacoste-Julien, Konstantina Palla, Alex Davies, Gjergji Kasneci, Thore Graepel, and Zoubin Ghahramani.  
1582 2013. SIGMa: Simple greedy matching for aligning large knowledge bases. In *SIGKDD*. 572–580.
- 1583 [112] Furong Li, Xin Luna Dong, Anno Langen, and Yang Li. 2017. Knowledge verification for LongTail verticals. *PVLDB*  
1584 10, 11 (2017).

- [113] Guoliang Li, Yudian Zheng, Ju Fan, Jiannan Wang, and Reynold Cheng. 2017. Crowdsourced data management: Overview and challenges. In *SIGMOD*. 1585–1586.
- [114] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. 2009. RiMOM: A dynamic multistrategy ontology alignment framework. *TKDE* 21, 8 (2009), 1218–1232. 1587–1588
- [115] Dionysios Logothetis, Soumyarupa De, and Kenneth Yocum. 2013. Scalable lineage capture for debugging DISC analytics. In *SoCC*. 17:1–17:15. 1589–1590
- [116] Yongtao Ma and Thanh Tran. 2013. TYPiMatch: Type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining (WSDM'13)*. 325–334. 1591–1592
- [117] Claire Mathieu, Ocan Sankur, and Warren Schudy. 2010. Online correlation clustering. In *STACS*. 573–584. 1594
- [118] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2000)*. 169–178. 1595–1596
- [119] W. P. McNeill, Hakan Kardes, and Andrew Borthwick. 2012. Dynamic record blocking: Efficient linking of massive databases in mapreduce. In *Proceedings of the 10th International Workshop on Quality in Databases (QDB'12)*. 1599
- [120] David G. McVitie and Leslie B. Wilson. 1970. Stable marriage assignment for unequal sets. *BIT Numerical Mathematics* 10, 3 (1970). 1600
- [121] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *INTERSPEECH*. 3771–3775. 1602–1603
- [122] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *SIGMOD*. 19–34. 1604–1605
- [123] Charini Nanayakkara, Peter Christen, and Thilina Ranbaduge. 2019. Robust temporal graph clustering for group record linkage. In *PAKDD*. 1607–1608
- [124] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection*. Morgan & Claypool. 1609
- [125] E. D. Nelson and J. R. Talburt. 2011. Entity resolution for longitudinal studies in education using OYSTER. In *IKE*. 1610
- [126] Markus Nentwig, Anika Groß, and Erhard Rahm. 2016. Holistic entity clustering for linked data. In *IEEE ICDM Workshops*. 194–201. 1611–1612
- [127] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. 2017. A survey of current Link Discovery frameworks. *Sem. Web* 8, 3 (2017), 419–436. 1613–1614
- [128] Axel-Cyrille Ngonga Ngomo and Sören Auer. 2011. LIMES—A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*. 1615
- [129] Maximilian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. In *NIPS*. 6338–6347. 1616–1617
- [130] Andriy Nikolov, Victoria S. Uren, Enrico Motta, and Anne N. De Roeck. 2008. Integration of semantically annotated data by the KnoFuss architecture. In *Proceedings of the 6th International Conference on Knowledge Engineering: Practice and Patterns (EKAW'08)*. 265–274. 1619–1620
- [131] Jordi Nin, Victor Muntés-Mulero, Norbert Martínez-Bazan, and Josep-Lluís Larriba-Pey. 2007. On the use of semantic blocking techniques for data cleansing and integration. In *Proceedings of the 11th International Database Engineering and Applications Symposium (IDEAS'07)*. 190–198. 1621–1622
- [132] Kevin O'Hare, Anna Jurek-Loughrey, and Cassio de Campos. 2019. A review of unsupervised and semi-supervised blocking methods for record linkage. In *Linking and Mining Heterogeneous and Multi-view Data*. Springer, 79–105. 1623–1624
- [133] George Papadakis, George Alexiou, George Papastefanatos, and Georgia Koutrika. 2015. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *PVLDB* 9, 4 (2015), 312–323. 1625–1626
- [134] George Papadakis, Konstantina Bereta, Themis Palpanas, and Manolis Koubarakis. 2017. Multi-core meta-blocking for big linked data. In *SEMANTICS*. 1627–1628
- [135] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. 2012. Beyond 100 million entities: Large-scale blocking-based resolution for heterogeneous data. In *The 5th International Conference on Web Search and Web Data Mining (WSDM'12)*. 53–62. 1629–1630
- [136] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederée, and Wolfgang Nejdl. 2013. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE TKDE* 25, 12 (2013), 2665–2682. 1631–1633
- [137] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. 2014. Meta-blocking: Taking entity resolution to the next level. *TKDE* 26, 8 (2014). 1634–1635
- [138] George Papadakis, George Papastefanatos, and Georgia Koutrika. 2014. Supervised meta-blocking. *PVLDB* 7, 14 (2014), 1929–1940. 1636–1637
- [139] George Papadakis, George Papastefanatos, Themis Palpanas, and Manolis Koubarakis. 2016. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *Proceedings of the 19th International Conference on Extending Database Technology (EDBT'16)*. 221–232. 1638–1641



- 1643 [140] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. A survey of blocking and  
1644 filtering techniques for entity resolution. *ACM Comput. Surv.* 53, 2 (2020), 31:1–31:42.
- 1645 [141] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative analysis of approximate  
1646 blocking techniques for entity resolution. *PVLDB* 9, 9 (2016), 684–695.
- 1647 [142] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, Nikiforos Pittaras, Giovanni Simonini, Dimitrios  
1648 Skoutas, Paul Isaris, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. 2020. JedAI<sup>3</sup>: Beyond batch,  
1649 blocking-based entity resolution. In *EDBT*. 603–606.
- 1650 [143] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. 2015. Progressive duplicate detection. *IEEE TKDE* 27, 5  
1651 (2015), 1316–1329.
- 1652 [144] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representa-  
1653 tion. In *EMNLP*. 1532–1543.
- 1654 [145] Banda Ramadan and Peter Christen. 2014. Forest-based dynamic sorted neighborhood indexing for real-time entity  
1655 resolution. In *CIKM*.
- 1656 [146] Banda Ramadan, Peter Christen, Huizhi Liang, and Ross W. Gayler. 2015. Dynamic sorted neighborhood indexing  
1657 for real-time entity resolution. *J. Data Inf. Quality* 6, 4, Article 15 (2015), 15:1–15:29.
- 1658 [147] Banda Ramadan, Peter Christen, Huizhi Liang, Ross W. Gayler, and David Hawking. 2013. Dynamic similarity-  
1659 aware inverted indexing for real-time entity resolution. In *Trends and Applications in Knowledge Discovery and Data  
1660 Mining—PAKDD International Workshops*. 47–58.
- 1661 [148] Vibhor Rastogi, Nilesh N. Dalvi, and Minos N. Garofalakis. 2011. Large-scale collective entity matching. *PVLDB* 4, 4  
1662 (2011), 208–218.
- 1663 [149] Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. 2017.  
1664 Snorkel: Rapid training data creation with weak supervision. *PVLDB* 11, 3 (2017), 269–282.
- 1665 [150] Orion Fausto Reyes-Galaviz, Witold Pedrycz, Ziyue He, and Nick J. Pizzi. 2017. A supervised gradient-based learning  
1666 algorithm for optimized entity resolution. *Data Knowl. Eng.* 112 (2017), 106–129.
- 1667 [151] Stephen V. Rice. 2007. Braided AVL trees for efficient event sets and ranked sets in the SIMSCRIPT III simulation  
1668 programming language. In *Proceedings of the MultiConference on Computer Simulation*. 150–155.
- 1669 [152] Shu Rong, Xing Niu, Evan Wei Xiang, Haofen Wang, Qiang Yang, and Yong Yu. 2012. A machine learning approach  
1670 for instance matching based on similarity metrics. In *Proceedings of the 11th International Semantic Web Conference  
1671 (ISWC’12)*. 460–475.
- 1672 [153] Alieh Saeedi, Markus Nentwig, Eric Peukert, and Erhard Rahm. 2018. Scalable matching and clustering of entities  
1673 with FAMER. *CSIMQ* 16 (2018), 61–83.
- 1674 [154] Alieh Saeedi, Eric Peukert, and Erhard Rahm. 2017. Comparative evaluation of distributed clustering schemes for  
1675 multi-source entity resolution. In *Proceedings of the 21st European Conference on Advances in Databases and Infor-  
1676 mation Systems (ADBIS’17)*. 278–293.
- 1677 [155] Alieh Saeedi, Eric Peukert, and Erhard Rahm. 2018. Using link features for entity clustering in knowledge graphs.  
1678 In *Proceedings of the 15th International Conference ESWC*. 576–592.
- 1679 [156] Murat Sariyar, Andreas Borg, and Klaus Pommerening. 2011. Controlling false match rates in record linkage using  
1680 extreme value theory. *Journal of Biomedical Informatics* 44, 4 (2011), 648–654.
- 1681 [157] Anish Das Sarma, Ankur Jain, Ashwin Machanavajjhala, and Philip Bohannon. 2012. An automatic blocking mech-  
1682 anism for large-scale de-duplication tasks. In *Proceedings of the 21st ACM International Conference on Information  
1683 and Knowledge Management (CIKM’12)*. 1055–1064.
- 1684 [158] Andrew T. Schneider, Arjun Mukherjee, and Eduard C. Dragut. 2018. Leveraging social media signals for record  
1685 linkage. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web (WWW’18)*. 1195–1204.
- 1686 [159] Chao Shao, Linmei Hu, Juan-Zi Li, Zhichun Wang, Tong Lee Chung, and Jun-Bo Xia. 2016. RiMOM-IM: A novel  
1687 iterative framework for instance matching. *J. Comput. Sci. Technol.* 31, 1 (2016), 185–197.
- 1688 [160] Liangcai Shu, Aiyou Chen, Ming Xiong, and Weiyi Meng. 2011. Efficient spectral neighborhood blocking for entity  
1689 resolution. In *Proceedings of the 27th International Conference on Data Engineering (ICDE’11)*. 1067–1078.
- 1690 [161] Giovanni Simonini, Sonia Bergamaschi, and H. V. Jagadish. 2016. BLAST: A loosely schema-aware meta-blocking  
1691 approach for entity resolution. *PVLDB* 9, 12 (2016), 1173–1184.
- 1692 [162] Giovanni Simonini, Luca Gagliardelli, Sonia Bergamaschi, and H. V. Jagadish. 2019. Scaling entity resolution: A  
1693 loosely schema-aware approach. *Inf. Syst.* 83 (2019), 145–165.
- 1694 [163] Giovanni Simonini, George Papadakis, Themis Palpanas, and Sonia Bergamaschi. 2019. Schema-agnostic progressive  
1695 entity resolution. *IEEE TKDE* 31, 6 (2019), 1208–1221.
- 1696 [164] Y. Sismanis, L. Wang, A. Fuxman, P. J. Haas, and B. Reinwald. 2009. Resolution-aware query answering for business  
1697 intelligence. In *Proceedings of the 25th International Conference on Data Engineering (ICDE’09)*. 976–987.
- 1698 [165] Dezhao Song and Jeff Heflin. 2011. Automatically generating data linkages using a domain-independent candidate  
1699 selection approach. In *Proceedings of the 10th International Semantic Web Conference (ISWC’11)*. 649–664.

## An Overview of End-to-End Entity Resolution for Big Data

127:41

- [166] Kostas Stefanidis, Vasilis Efthymiou, Melanie Herschel, and Vassilis Christophides. 2014. Entity resolution in the web of data. In *Proceedings of the 23rd International World Wide Web Conference (WWW'14), Companion Volume*. 203–204. 1700–1702
- [167] Rebecca C. Steorts, Samuel L. Ventura, Mauricio Sadinle, and Stephen E. Fienberg. 2014. A comparison of blocking methods for record linkage. In *Proceedings of the 2014 International Conference on Privacy in Statistical Databases (PSD'14)*. 253–268. 1703–1705
- [168] Weifeng Su, Jiying Wang, and Frederick H. Lochovsky. 2010. Record matching over query results from multiple web databases. *IEEE TKDE* 22, 4 (2010), 578–589. 1706–1707
- [169] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. 2011. PARIS: Probabilistic alignment of relations, instances, and schema. *PVLDB* 5, 3 (2011), 157–168. 1708–1709
- [170] Zequn Sun, Wei Hu, and Chengkai Li. 2017. Cross-lingual entity alignment via joint attribute-preserving embedding. In *Proceedings of the 16th International Semantic Web Conference (ISWC'17)*. 628–644. 1710–1711
- [171] Zequn Sun, Wei Hu, Qingheng Zhang, and Yuzhong Qu. 2018. Bootstrapping entity alignment with knowledge graph embedding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. 4396–4402. 1712–1713
- [172] Zequn Sun, Qingheng Zhang, Wei Hu, Chengming Wang, Muhao Chen, Farahnaz Akrami, and Chengkai Li. 2020. A benchmarking study of embedding-based entity alignment for knowledge graphs. *CoRR* abs/2003.07743. 1714–1715
- [173] Saravanan Thirumuruganathan, Shameem A. Puthiya Parambath, Mourad Ouzzani, Nan Tang, and Shafiq Joty. 2018. Reuse and adaptation for entity resolution through transfer learning. *CoRR* abs/1809.11084. 1716–1717
- [174] Bayu Distiawan Trisedya, Jianzhong Qi, and Rui Zhang. 2019. Entity alignment between knowledge graphs using attribute embeddings. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*. 297–304. 1718–1720
- [175] Stijn Marinus Van Dongen. 2000. *Graph Clustering by Flow Simulation*. Ph.D. Dissertation. Utrecht University. 1721
- [176] Marieke van Erp, Pablo N. Mendes, Heiko Paulheim, Filip Ilievski, Julien Plu, Giuseppe Rizzo, and Jörg Waitelonis. 2016. Evaluating entity linking: An analysis of current benchmark datasets and a roadmap for doing a better job. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC'16)*. 1722–1724
- [177] Vasilis Verroios and Hector Garcia-Molina. 2015. Entity resolution with crowd errors. In *Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE'15)*. 219–230. 1725–1726
- [178] Vasilis Verroios, Hector Garcia-Molina, and Yannis Papakonstantinou. 2017. Waldo: An adaptive human interface for crowd entity resolution. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD'17)*. 1133–1148. 1727–1729
- [179] Norases Vesdapunt, Kedar Bellare, and Nilesh N. Dalvi. 2014. Crowdsourcing algorithms for entity resolution. *PVLDB* 7, 12 (2014), 1071–1082. 1730–1731
- [180] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. 2009. Silk—A link discovery framework for the web of data. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web (LDOW'09)*. 1732–1733
- [181] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing entity resolution. *PVLDB* 5, 11 (2012), 1483–1494. 1734–1735
- [182] Jiannan Wang, Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. 2014. A sample-and-clean framework for fast and accurate query processing on dirty data. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*. 469–480. 1736–1738
- [183] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2013. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*. 229–240. 1739–1741
- [184] Jiannan Wang, Guoliang Li, Jeffrey Xu Yu, and Jianhua Feng. 2011. Entity matching: How similar is similar. *PVLDB* 4, 10 (2011), 622–633. 1742–1743
- [185] Sibo Wang, Xiaokui Xiao, and Chun-Hee Lee. 2015. Crowd-based deduplication: An adaptive approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*. 1263–1277. 1744–1745
- [186] Xiaolan Wang, Laura M. Haas, and Alexandra Meliou. 2018. Explaining data integration. *IEEE Data Eng. Bull.* 41, 2 (2018), 47–58. 1746–1747
- [187] Yihan Wang, Shaoxu Song, Lei Chen, Jeffrey Xu Yu, and Hong Cheng. 2017. Discovering conditional matching rules. *TKDD* 11, 4 (2017), 46:1–46:38. 1748–1749
- [188] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. 2018. Cross-lingual knowledge graph alignment via graph convolutional networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP'18)*. 349–357. 1750–1752
- [189] Melanie Weis and Felix Naumann. 2004. Detecting duplicate objects in XML documents. In *Proceedings of the International Workshop on Information Quality in Information Systems (IQIS'04)*. 10–19. 1753–1754
- [190] Melanie Weis and Felix Naumann. 2006. Detecting duplicates in complex XML data. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*. 109. 1755–1756

- 1757 [191] Michael J. Welch, Aamod Sane, and Chris Drome. 2012. Fast and accurate incremental entity resolution relative to  
 1758 an entity knowledge base. In *Proceedings of the 21st ACM International Conference on Information and Knowledge*  
 1759 *Management (CIKM'12)*. 2667–2670.
- 1760 [192] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. 2013. Question selection for crowd entity resolu-  
 1761 tion. *PVLDB* 6, 6 (2013), 349–360.
- 1762 [193] Steven Euijong Whang, David Marmaros, and Hector Garcia-Molina. 2013. Pay-as-you-go entity resolution. *IEEE*  
 1763 *TKDE* 25, 5 (2013), 1111–1124.
- 1764 [194] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. 2009. Entity resolution with iterative  
 1765 blocking. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD'09)*.  
 1766 219–232.
- 1767 [195] Derry Tanti Wijaya and Stéphane Bressan. 2009. Ricochet: A family of unconstrained algorithms for graph cluster-  
 1768 ing. In *Proceedings of the 14th International Conference on Database Systems for Advanced Applications (DASFAA'09)*.  
 1769 153–167.
- 1770 [196] Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural  
 1771 networks. *Neural Computation* 1, 2 (1989), 270–280.
- 1772 [197] Vijaya Krishna Yalavarthi, Xiangyu Ke, and Arijit Khan. 2017. Select your questions wisely: For entity resolution with  
 1773 crowd errors. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM'17)*.  
 1774 317–326.
- 1775 [198] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. 2006. *Similarity Search—The Metric Space Ap-*  
 1776 *proach*. Kluwer.
- 1777 [199] Chen Jason Zhang, Rui Meng, Lei Chen, and Feida Zhu. 2015. CrowdLink: An error-tolerant model for linking  
 1778 complex records. In *Proceedings of the 2nd International Workshop on Exploratory Search in Databases and the Web*  
 1779 *(ExploreDB'15)*. 15–20.
- 1780 [200] Fulin Zhang, Zhipeng Gao, and Kun Niu. 2017. A pruning algorithm for Meta-blocking based on cumulative weight.  
 1781 In *JPCS*, Vol. 887.
- 1782 [201] Qingheng Zhang, Zequn Sun, Wei Hu, Muhao Chen, Lingbing Guo, and Yuzhong Qu. 2019. Multi-view knowledge  
 1783 graph embedding for entity alignment. In *Proceedings of the 28th International Joint Conference on Artificial Intelli-*  
 1784 *gence (IJCAI'19)*. 5429–5435.
- 1785 [202] Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and David Page. 2020. AutoBlock: A  
 1786 hands-off blocking framework for entity matching. In *The 13th ACM International Conference on Web Search and*  
 1787 *Data Mining (WSDM'20)*. 744–752.
- 1788 [203] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end fuzzy entity-matching using pre-trained deep models and  
 1789 transfer learning. In *The World Wide Web Conference (WWW'19)*. 2413–2424.
- 1790 [204] Qibin Zheng, Xingchun Diao, Jianjun Cao, Xiaolei Zhou, Yi Liu, and Hongmei Li. 2018. Multi-modal space structure:  
 1791 A new kind of latent correlation for multi-modal entity resolution. *CoRR* abs/1804.08010.
- 1792 [205] Hao Zhu, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2017. Iterative entity alignment via joint knowledge em-  
 1793 beddings. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. 4258–4264.

1794 Received April 2020; revised July 2020; accepted August 2020

### **Author Queries**

- Q1:** AU: Please supply the CCS Concepts 2012 codes per the ACM style indicated on the ACM website. Please include the CCS Concepts XML coding as well.
- Q2:** AU: Please provide current mailing addresses, including street, for all authors.
- Q3:** AU: Please check the sentence beginning “Similarity evidence of entities...” for clarity.
- Q4:** AU: Please define NLP.
- Q5:** AU: Please define LSH.