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

VASSILIS CHRISTOPHIDES, ENSEA, ETIS Lab, France
VASILIS EFTHYMIOU, IBM Research, USA
THEMIS PALPANAS, Universite de Paris & 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.

ACM Reference Format:
Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 1, 1, Article 1 (January 2020), 37 pages. https://doi.org/10.1145/3418896

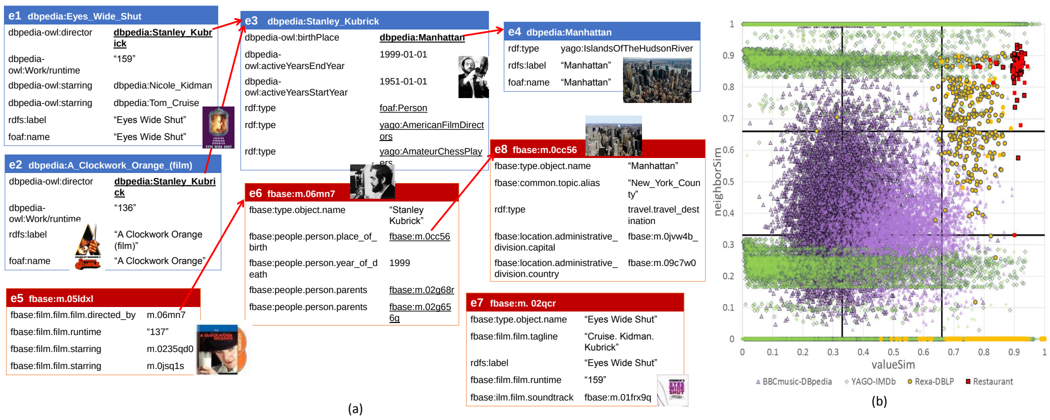
1 INTRODUCTION

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).

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 (KB) 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).

Authors' addresses: Vassilis Christophides, ENSEA, ETIS Lab, France, vassilis.christophides@inria.fr; Vasilis Efthymiou, IBM Research, USA, vasilis.efthymiou@ibm.com; Themis Palpanas, Universite de Paris & French University Institute (IUF), France, themis@mi.parisdescartes.fr; George Papadakis, National and Kapodistrian University of Athens, Greece, gpapadis@di.uoa.gr; Kostas Stefanidis, Tampere University, Finland, konstantinos.stefanidis@tuni.fi.

0360-0300/2020/1-ART1 $15.00
https://doi.org/10.1145/3418896



e1 dbpedia:Eyes_Wide_Shut	
dbpedia-owl:director	**dbpedia:Stanley_Kubrick**
dbpedia-owl:Work/runtime	"159"
dbpedia-owl:starring	dbpedia:Nicole_Kidman
dbpedia-owl:starring	dbpedia:Tom_Cruise
rdfs:label	"Eyes Wide Shut"
foaf:name	"Eyes Wide Shut"

e2 dbpedia:A_Clockwork_Orange_(film)	
dbpedia-owl:director	**dbpedia:Stanley_Kubrick**
dbpedia-owl:Work/runtime	"136"
rdfs:label	"A Clockwork Orange (film)"
foaf:name	"A Clockwork Orange"

e5 fbase:m.05ldxl	
fbase:film.film.film.directed_by	m.06mn7
fbase:film.film.runtime	"137"
fbase:film.film.starring	m.0235qd0
fbase:film.film.starring	m.0jsq1s

e3 dbpedia:Stanley_Kubrick	
dbpedia-owl:birthPlace	**dbpedia:Manhattan**
dbpedia-owl:activeYearsEndYear	1999-01-01
dbpedia-owl:activeYearsStartYear	1951-01-01
rdf:type	foaf:Person
rdf:type	yago:AmericanFilmDirectors
rdf:type	yago:AmateurChessPlayers

e6 fbase:m.06mn7	
fbase:type.object.name	"Stanley Kubrick"
fbase:people.person.place_of_birth	fbase:m.0cc56
fbase:people.person.year_of_death	1999
fbase:people.person.parents	fbase:m.02g68r
fbase:people.person.parents	fbase:m.02g656g

e4 dbpedia:Manhattan	
rdf:type	yago:IslandsOfTheHudsonRiver
rdfs:label	"Manhattan"
foaf:name	"Manhattan"

e8 fbase:m.0cc56	
fbase:type.object.name	"Manhattan"
fbase:common.topic.alias	"New_York_County"
rdf:type	travel.travel_destination
fbase:location.administrative_division.capital	fbase:m.0jvw4b_
fbase:location.administrative_division.country	fbase:m.09c7w0

e7 fbase:m. 02qcr	
fbase:type.object.name	"Eyes Wide Shut"
fbase:film.film.tagline	"Cruise. Kidman. Kubrick"
rdfs:label	"Eyes Wide Shut"
fbase:film.film.runtime	"159"
fbase:ilm.film.soundtrack	fbase:m.01frx9q

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, resp. (b) Value and neighbor similarity distribution of matches in four datasets.

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 an x100 growth since its first edition) in 10 domains with >200B triples (i.e., $< subject, predicate, object >$) describing more than 60M entities of different types[1]; the life-science domain alone accounts for >350 datasets.
- *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 one KB[2].
- *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

[1]https://lod-cloud.net
[2]https://lov.linkeddata.es/dataset/lov

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[3].

- *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_5) 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[4], Rexa-DBLP[5], BBCmusic-DBpedia[6] and YAGO-IMDb[7]. Every dot corresponds to a different matching pair, while its shape denotes the respective dataset. The horizontal axis reports the normalized value similarity based on the common words in a pair of descriptions (weighted Jaccard [111]), while the vertical one reports the maximum value similarity of their respective entity neighbors. We can observe that the value-based similarity of matching entities significantly varies across different datasets. For *strongly similar entities* (e.g., value similarity > 0.5), existing duplicate detection techniques work well, but to resolve *nearly similar entities* (e.g., value similarity < 0.5), we need advanced ways of exploiting evidence about the similarity of neighboring entities, due to the Variety in entity types.

Additional challenges are raised by the *Velocity* of Big Data Entities. Even though ER is historically framed as an offline task that improves data quality in data warehouses upon completion of data

[3]http://km.aifb.kit.edu/projects/dyldo
[4]http://oaei.ontologymatching.org/2010/im
[5]http://oaei.ontologymatching.org/2009/instances
[6]http://datahub.io/dataset/bbc-music, http://km.aifb.kit.edu/projects/btc-2012
[7]http://www.yago-knowledge.org, http://www.imdb.com

integration, many services now require to *resolve entities in real-time*. Such services strive for incremental ER workflows over *dynamic sources* that can sacrifice completeness of the resulting matches as long as *query-based* [5, 17] or *streaming* [96] execution strategies can be supported.

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

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

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

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

2 ER PROCESSING TASKS AND WORKFLOWS

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

(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.
(2) *Dirty ER*, where the goal is to identify the duplicates within a single entity collection.
(3) *Multi-source ER*, when more than two entity collections are given as input.

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

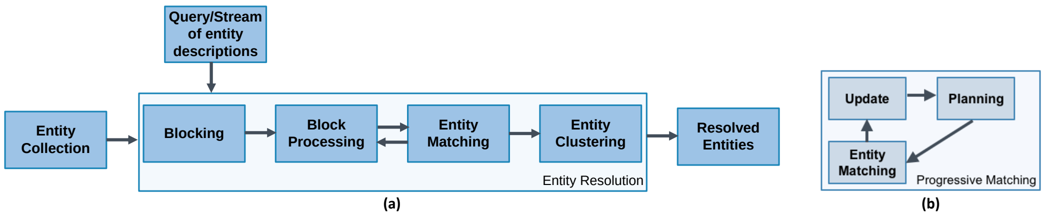


Fig. 2. (a) The generic end-to-end workflow for Entity Resolution. (b) Budget-aware Matching.

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.

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.

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.

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.

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 $R = \{r_1, r_2, \ldots, r_m\}$, such that: (i) $\forall e_i,\ e_j \in r_k\ M(e_i, e_j) = true$, (ii) $\forall e_i \in r_k \forall e_j \in r_l\ M(e_i, e_j) = false$, and (iii) $\cup_{r_i} r_i \in R = \mathcal{E}$, where $\mathcal{E}$ stands for the input entity collection. This partitioning corresponds to the resulting set of resolved entities in Figure 2(a).

Figure 2(b) illustrates the additional processing tasks that are required when an ER workflow is subject to budget restrictions in terms of time or number of comparisons. These restrictions essentially call for an approximate solution to ER, as an indirect way of addressing Volume. Rather than finding all entity matches, the goal of *budget-aware ER* is to progressively identify as many matches as possible within a specified cost budget. It extends batch, budget-agnostic ER workflows with a *Planning* and *Update* phase that typically work on windows [2]. Planning is responsible for selecting *which* pairs of descriptions will be compared for matching and in *what order*, based on the cost/benefit trade-off. Within every window, it essentially favors the more promising comparisons, which are more likely to increase the targeted benefit (e.g., the number of matches) in the remaining

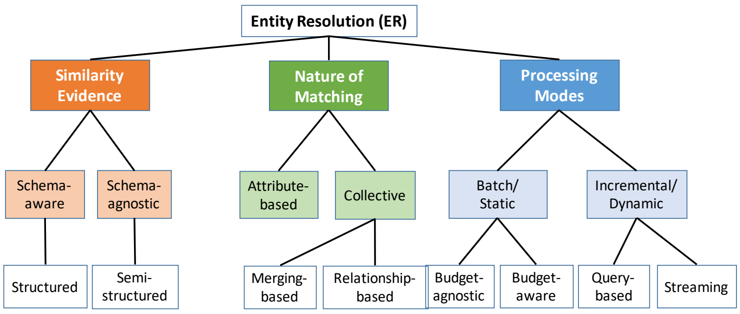


Fig. 3. Taxonomy of ER settings and approaches.

budget. Those comparisons are performed first in the current window and thus, a higher benefit is achieved as early as possible. The Update phase takes into account the results of Matching, such that Planning in a subsequent window will promote the comparison of pairs influenced by the previous matches. This iterative ER process continues until the budget is exhausted. Both phases rely on a graph of dependencies among descriptions [48], which leverages budget-agnostic blocking methods.

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

Taxonomy of ER settings and approaches. Overall, Figure 3 illustrates the taxonomy of ER settings based on the key characteristics. Blocking, Matching and Clustering methods that operate on relational data are *schema-aware*, as opposed to the *schema-agnostic* methods, which are more flexible regarding the structure, since they consider all attribute values. In the context of Big Data, nearly similar entities are resolved by going beyond *attribute-based* ER techniques, which examine each pair of descriptions independently from other pairs. To match graph-based descriptions 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 trade-off 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"*.

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

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

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

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

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

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

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

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

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 [130]	✓			✓
Token Blocking [135]	✓		✓	
Attribute Clustering Blocking [135]	✓		✓	
Prefix-Infix(-Suffix) Blocking [134]	✓		✓	
Suffix Arrays Blocking [1]	✓		✓	
Extended Suffix Arrays Blocking [34,132]	✓		✓	
Q-Grams Blocking [34,132]	✓		✓	
Extended Q-Grams Blocking [34,132]	✓		✓	
Canopy Clustering [34,117]	✓			✓
Extended Canopy Clustering [34,132]	✓			✓
Sorted Neighborhood [83]	✓			✓
Extended Sorted Neighborhood [83]	✓		✓	
Hetero [99]		✓	✓	
Extended DNF BSL [100]		✓	✓	

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.

4 BLOCK PROCESSING

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

4.1 Block Cleaning

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

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

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

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

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

4.2 Comparison Cleaning

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

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

For edge pruning, the following algorithms are available: *Weighted Edge Pruning* [137] removes all edges that do not exceed the average edge weight; *Cardinality Edge Pruning* retains the globally K top weighted edges [137, 200]; *Weighted Node Pruning* (WNP) [137] and *BLAST* [161] retain in each node neighborhood the descriptions that exceed a local threshold; *Cardinality Node Pruning* (CNP) retains the top-k weighted edges in each node neighborhood [137]; *Reciprocal WNP* and *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 labelling effort, by selecting a very small training set that maintains high effectiveness.

Parellelization. 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

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 [135,134]	✓			✓	✓	
Block Filtering [138,140]	✓			✓	✓	
Rollup Canopies [156]	✓			✓		✓
Size-based Block Clustering [64]	✓			✓	✓	
Iterative Blocking [193]	✓		✓		✓	
Block Scheduling [135,162]	✓		✓		✓	
Block Pruning [135]	✓		✓		✓	
Weighted Edge Pruning [136]		✓		✓	✓	
Cardinality Edge Pruning [136,199]		✓		✓	✓	
(Reciprocal) Weighted Node Pruning [136,138]		✓		✓	✓	
BLAST [160]		✓		✓	✓	
(Reciprocal) Cardinality Node Pruning [136,138]		✓		✓	✓	
Disjunctive Blocking Graph [55]		✓		✓	✓	
Transitive LSH [166]		✓		✓	✓	
SPAN [159]		✓		✓	✓	
Comparison Propagation [135]		✓		✓	✓	
Supervised Meta-blocking [137]		✓		✓		✓
BLOSS [17]		✓		✓		✓

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

5 MATCHING

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

5.1 Preliminaries

The matching decision is typically made by a match function M, which 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 $M(e_i, e_j) = false$ meaning that e_i and e_j are not matches.

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

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

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

Finding a similarity metric which can perfectly distinguish all matches from non-matches using simple pairwise comparisons on the attribute values of two descriptions is practically impossible. In particular, similarity metrics are too restrictive to identify nearly similar matches. Thus, in reality, we seek similarity functions that will be only good enough, i.e., minimize the number of misclassified pairs, and rely on collective ER approaches to propagate the similarity of the entity neighbors of two descriptions to the similarity of those descriptions. In this inherently iterative process, the employed match function is based on a similarity that dynamically changes from iteration to iteration, and its results may include a third state, the *uncertain* one. Specifically, given two similarity thresholds θ and θ', with $\theta' < \theta$, the match function at iteration n, M^n, is given by:

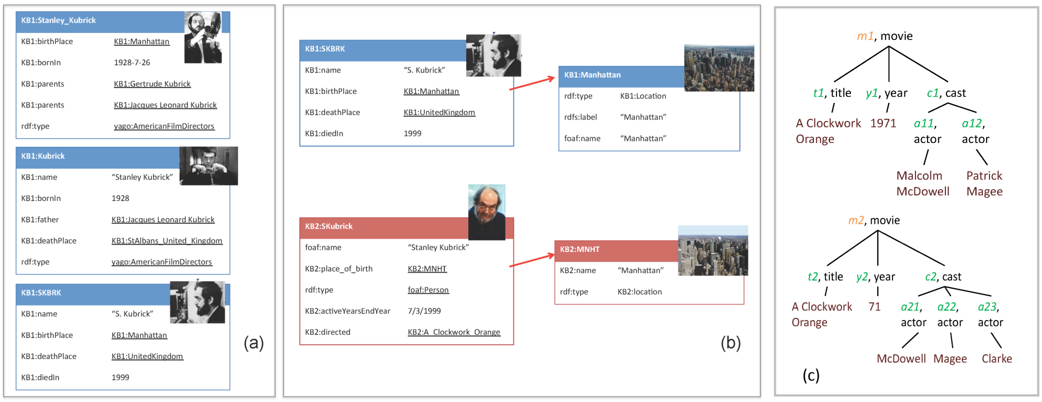
$$M^n(e_i, e_j) = \begin{cases} \text{true, if } sim^{n-1}(e_i, e_j) \geq \theta, \\ \text{false, if } sim^{n-1}(e_i, e_j) \leq \theta', \\ \text{uncertain, otherwise.} \end{cases}$$

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.

5.2 Collective methods

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.

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	
KB1:birthPlace	KB1:Manhattan
KB1:bornIn	1928-7-26
KB1:parents	KB1:Gertrude Kubrick
KB1:parents	KB1:Jacques Leonard Kubrick
rdf:type	yago:AmericanFilmDirectors

KB1:Kubrick	
KB1:name	"Stanley Kubrick"
KB1:bornIn	1928
KB1:father	KB1:Jacques Leonard Kubrick
KB1:deathPlace	KB1:StAlbans_United_Kingdom
rdf:type	yago:AmericanFilmDirectors

KB1:SKBRK	
KB1:name	"S. Kubrick"
KB1:birthPlace	KB1:Manhattan
KB1:deathPlace	KB1:UnitedKingdom
KB1:diedIn	1999

(a)

KB1:SKBRK	
KB1:name	"S. Kubrick"
KB1:birthPlace	KB1:Manhattan
KB1:deathPlace	KB1:UnitedKingdom
KB1:diedIn	1999

KB1:Manhattan	
rdf:type	KB1:Location
rdfs:label	"Manhattan"
foaf:name	"Manhattan"

KB2:SKubrick	
foaf:name	"Stanley Kubrick"
KB2:place_of_birth	KB2:MNHT
rdf:type	foaf:Person
KB2:activeYearsEndYear	7/3/1999
KB2:directed	KB2:A_Clockwork_Orange

KB2:MNHT	
KB2:name	"Manhattan"
rdf:type	KB2:location

(b)

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.

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

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

5.2.1 Schema-aware methods. In *merging-based collective ER*, the matching decision between two descriptions triggers a merge operation, which transforms the initial entity collection by adding the new, merged description and potentially removing the two initial descriptions. This change also triggers more updates in the matching decisions, since the new, merged description needs to be compared to the other descriptions of the collection. Intuitively, the final result of merging-based collective ER is a new entity collection, which is the result of merging all the matches found in the initial collection. In other words, there should be an 1-1 correspondence between the descriptions in the resolution results and the actual real-world entities from the input entity collection.

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

- *Idempotence:* $\forall e_i, M(e_i, e_i) = true$ and $\mu(e_i, e_i)=e_i$
- *Commutativity:* $\forall e_i, e_j, M(e_i, e_j)=true \Leftrightarrow M(e_j, e_i)= true$ and $\mu(e_i, e_j) = \mu(e_j, e_i)$
- *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, $\mu(e_i, \mu(e_j, e_k))=\mu(\mu(e_i, e_j), e_k)$
- *Representativity:* If $e_k = \mu(e_i, e_j)$, then for any e_l such that $M(e_i, e_l) = true$, $M(e_k, e_l) = true$

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

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

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

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

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

where sim can be a relational similarity function and θ 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.

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

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

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

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

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

5.3 Learning-based methods

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

Supervised Learning. *Adaptive Matching* [41] learns from the training data a composite function that combines many attribute similarity measures. Similarly, *MARLIN* [20] uses labeled data at 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 attribute values. In the context of Clean-Clean ER, having generated the positive example ($e1$, $e2$), where $e1$ belongs to entity collection $\mathcal{E}_1$ and $e2$ to $\mathcal{E}_2$, for every other positive example ($e3$, $e4$), where $e3 \in \mathcal{E}_1$ and $e4 \in \mathcal{E}_2$, it further infers the negative examples ($e1$, $e4$) and ($e3$, $e2$). The resulting training set is first used by the system for Schema Matching to align the attributes in the input datasets. The attribute alignment and the training sets are then used to simultaneously learn two functions, one for Blocking and the other for Matching.

5.4 Parallel methods

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

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

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

Table 3. Taxonomy of the Matching methods discussed in Section 5. MB stands for Merging-based, RB for Relationship-based, S for Supervised, WS for Weakly Supervised and U for Unsupervised Learning.

	Schema awareness		Nature of comparisons		Algorithmic foundations	
	Schema-aware	Schema-agnostic	Attribute-based	Collective	Learning-based	Non-learning
Swoosh [14]	✓			MB		✓
D-Swoosh [13]	✓			MB		✓
Collective ER [15]	✓			RB		✓
Hybrid Collective ER [47]	✓			MB,RB		✓
Collective ER for XML [189]		✓		RB		✓
SiGMa [110]		✓		RB		✓
LINDA [20]		✓		RB		✓
RiMOM [113, 158]		✓		RB		✓
PARIS [168]		✓		RB		✓
MinoanER [55]		✓		RB		✓
Adaptive Matching [40]	✓		✓		S	
MARLIN [19]	✓		✓		S	
Gradient-based Matching [149]	✓		✓		S	
BN-based Collective ER [88]		✓		RB	S	
GenLink [90]		✓	✓		S	
Transfer learning [172]	✓		✓		WS	
Transfer learning for RDF [151]		✓		RB	WS	
Unsupervised ensemble [93]	✓		✓		U	
Unsupervised ER for RDF [101]	✓		✓		U	
Large-scale Collective ER [147]	✓			RB		✓

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

5.5 Discussion

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

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

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

6 CLUSTERING METHODS

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

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

1) For Clean-Clean ER, clustering typically relies on the 1-1 correspondence between the input data sources. The most popular technique is *Unique Mapping Clustering* [21, 111], which first sorts all edges in decreasing weight. At each iteration, the top edge is considered a match, if none of the adjacent descriptions has already been matched. The process ends when the top edge has a similarity lower than a threshold t. Essentially, this approach provides an efficient solution to the *Stable Marriage* problem for unequal sets [120], given that Clean-Clean ER forms a (usually

unbalanced) bipartite similarity graph. The *Hungarian algorithm* is also applicable, though at a much higher computational cost, unless an approximation is used, as in [46, 108].

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

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

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

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

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

7.1 Schema-aware methods

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

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

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

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

A static, block-centric method is the *Hierarchy of Record Partitions* (**HRP**) [193], which presumes that the distance of two records can be naturally estimated through a certain attribute (e.g., product price). Essentially, it builds a hierarchy of blocks, such that the matching likelihood of two descriptions is proportional to the level in which they co-occur for the first time: the blocks at the bottom of the hierarchy contain the descriptions with the highest matching likelihood, and vice 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 $\frac{fr_{j,i}}{fr_i + fr_j - fr_{i,j}}$, 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 $< e_i, e_j >$ 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.

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) [192]	✓			✓		✓		✓	
Dynamic PSN [142]	✓			✓		✓			✓
Progressive Blocking [142]	✓		✓		✓				✓
Hierarchy of Record Partitions [192]	✓		✓		✓			✓	
Ordered List of Records [192]	✓		✓				✓	✓	
Progressive Relational Entity Resolution [2]	✓		✓			✓			✓
Local Schema-agnostic PSN [162]		✓		✓		✓		✓	
Global Schema-agnostic PSN [162]		✓		✓		✓		✓	
Progressive Block Scheduling [162]		✓	✓		✓	✓		✓	
Progressive Profile Scheduling [162]		✓	✓			✓	✓	✓	

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

7.3 Discussion

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

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

8 INCREMENTAL ER

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

As an example, consider an application resolving the entities described across news feeds, which arrive in a streaming fashion [9, 19, 96]. A journalist using this application could be provided with several facts regarding a breaking news story (e.g., persons, buildings, services affected by an earthquake), as they get published by different agencies or witnesses, enabling her/him to form a 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 to the complexity of the collective ER strategy, this approach cannot provide real-time answers for large datasets.

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

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

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

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

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

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

8.3 Dynamic Clustering

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

8.4 Discussion

Table 5 organizes all methods discussed in this section into a taxonomy formed by three criteria: the ER workflow task corresponding to each method, its schema-awareness and its algorithmic foundation (learning-based or non-learning). These works are crafted for resolving entities in

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 [35]	✓			✓			✓
DySimII [146]	✓			✓			✓
F-DySNI [144,145]	✓			✓			✓
SBlockSketch [95]	✓	✓		✓			✓
Query-driven Collective ER [16]		✓		✓			✓
Query-driven ER with uncertainty [83]		✓		✓			✓
UDD [167]		✓		✓		Unsupervised	
Sample-and-clean [181]		✓		✓			✓
QuERy [5]		✓		✓			✓
QDA [4]		✓		✓			✓
Adaptive Product Normalization [18]		✓		✓		Supervised	
Incremental Correlation Clustering [76]			✓	✓			✓

(near) real time, not necessarily covering the whole input entity collections, but only a subset 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 towards 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.

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

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

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

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

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

Methods leveraging additional types of features to refine relation-based embeddings include the following. *AttrE* [174] uses character-level literal embeddings over a unified vector space for the 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?
Challenge 2: How should HITs be formulated?
Challenge 3: How can we maximize accuracy, while minimizing the overall monetary cost?
Challenge 4: How can we restrict the labour cost?

Below, we examine the main solutions to each challenge.

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.

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 trade-off 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 with overwhelming information, i.e., with complex structures and attributes. A probabilistic framework then selects the k best attributes.

Challenge 3: To optimize the trade-off between accuracy and monetary cost, the transitive relation is typically leveraged; if the relation between two descriptions can be inferred by transitivity from the already detected duplicates, it is not crowd-sourced. This inference takes two flavours [28]: *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* 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 approaches [64, 98, 179, 183, 192] that minimize the number of HITs submitted to workers, reducing significantly the crowd-sourcing overhead. Their key insight is that finding matches before non-matches accelerates the ER process, by making the most of the transitive closure.

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

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

9.3 Rule-based ER methods

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

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

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

Table 6. The main open-source ER Tools (a feature in parenthesis is partially supported).

Tool	Blocking	Block Processing	Matching	Clustering	Parallelization	Bugdet-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

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 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][8] 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 3 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 matching.

[8]http://pages.cs.wisc.edu/~anhai/papers/magellan-tr.pdf

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

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

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

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

9.6 Discussion

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

[9]https://github.com/larsga/Duke

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 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 such cases will be helpful for ER, since knowledge about long tail entities is less accessible, not redundant and hard to obtain.

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

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

11 CONCLUSIONS

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

Acknowledgements. This work was partially funded by the EU H2020 project ExtremeEarth (825258).

REFERENCES

[1] A. N. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI*, 2005.
[2] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11), 2014.
[3] Y. Altowim and S. Mehrotra. Parallel progressive approach to entity resolution using mapreduce. In *ICDE*, pages 909–920, 2017.
[4] H. Altwaijry, D. V. Kalashnikov, and S. Mehrotra. QDA: A query-driven approach to entity resolution. *TKDE*, 29(2), 2017.
[5] H. Altwaijry, S. Mehrotra, and D. V. Kalashnikov. Query: A framework for integrating entity resolution with query processing. *PVLDB*, 9(3):120–131, 2015.
[6] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.
[7] R. Angell, B. Johnson, Y. Brun, and A. Meliou. Themis: automatically testing software for discrimination. In *ESEC/FSE*, 2018.
[8] S. Araújo, D. T. Tran, A. P. de Vries, and D. Schwabe. SERIMI: class-based matching for instance matching across heterogeneous datasets. *TKDE*, 27(5):1397–1410, 2015.
[9] T. B. Araújo, K. Stefanidis, C. E. S. Pires, J. Nummenmaa, and T. P. da Nóbrega. Schema-agnostic blocking for streaming data. In *ACM SAC*, pages 412–419, 2020.
[10] J. A. Aslam, E. Pelekhov, and D. Rus. The star clustering algorithm for static and dynamic information organization. *J. Graph Algorithms Appl.*, 8:95–129, 2004.
[11] T. Baltrusaitis, C. Ahuja, and L.-P. Morency. Challenges and applications in multimodal machine learning. In *The Handbook of Multimodal-Multisensor Interfaces*, pages 17–48. ACM and Morgan & Claypool, 2019.
[12] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
[13] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *JMLR*, 3:1137–1155, 2003.
[14] O. Benjelloun, H. Garcia-Molina, H. Gong, H. Kawai, T. E. Larson, D. Menestrina, and S. Thavisomboon. D-swoosh: A family of algorithms for generic, distributed entity resolution. In *ICDCS*, page 37, 2007.
[15] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *VLDB J.*, 18(1):255–276, 2009.
[16] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *TKDD*, 1(1):5, 2007.
[17] I. Bhattacharya and L. Getoor. Query-time entity resolution. *J. Artif. Intell. Res.*, 30:621–657, 2007.
[18] G. D. Bianco, M. A. Gonçalves, and D. Duarte. BLOSS: effective meta-blocking with almost no effort. *Inf. Syst.*, 75:75–89, 2018.
[19] M. Bilenko, S. Basu, and M. Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *ICDM*, pages 58–65, 2005.
[20] M. Bilenko and R. J. Mooney. Adaptive Duplicate Detection using Learnable String Similarity Measures. In *SIGKDD*, 2003.

[21] C. Böhm, G. de Melo, F. Naumann, and G. Weikum. LINDA: distributed web-of-data-scale entity matching. In *CIKM*, 2012.
[22] U. Brunner and K. Stockinger. Entity matching with transformer architectures - A step forward in data integration. pages 463–473, 2020.
[23] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD*, 2016.
[24] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. A partial-order-based framework for cost-effective crowdsourced entity resolution. *VLDB J.*, 27(6):745–770, 2018.
[25] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.
[26] M. Chen, Y. Tian, K. Chang, S. Skiena, and C. Zaniolo. Co-training embeddings of knowledge graphs and entity descriptions for cross-lingual entity alignment. In *IJCAI*, 2018.
[27] M. Chen, Y. Tian, M. Yang, and C. Zaniolo. Multilingual knowledge graph embeddings for cross-lingual knowledge alignment. In *IJCAI*, 2017.
[28] X. Chen. Crowdsourcing entity resolution: a short overview and open issues. In *GvDB*, pages 72–77, 2015.
[29] X. Chen, E. Schallehn, and G. Saake. Cloud-scale entity resolution: Current state and open challenges. *OJBD*, 4(1), 2018.
[30] Y. Chiang, A. Doan, and J. F. Naughton. Modeling entity evolution for temporal record matching. In *SIGMOD*, pages 1175–1186, 2014.
[31] Y. Chiang, A. Doan, and J. F. Naughton. Tracking entities in the dynamic world: A fast algorithm for matching temporal records. *PVLDB*, 7(6):469–480, 2014.
[32] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.
[33] P. Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *KDD*, pages 1065–1068, 2008.
[34] P. Christen. *Data Matching*. Springer, 2012.
[35] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE*, 24(9):1537–1555, 2012.
[36] P. Christen, R. W. Gayler, and D. Hawking. Similarity-aware indexing for real-time entity resolution. In *CIKM*, pages 1565–1568, 2009.
[37] V. Christophides, V. Efthymiou, and K. Stefanidis. *Entity Resolution in the Web of Data*. Morgan & Claypool, 2015.
[38] X. Chu, I. F. Ilyas, and P. Koutris. Distributed data deduplication. *PVLDB*, 9(11):864–875, 2016.
[39] Y. Chung, T. Kraska, N. Polyzotis, K. Tae, and S. E. Whang. Slice finder: Automated data slicing for model validation. In *ICDE*, 2019.
[40] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.
[41] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *SIGKDD*, 2002.
[42] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, pages 1431–1446, 2017.
[43] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
[44] J. Debattista, C. Lange, S. Auer, and D. Cortis. Evaluating the quality of the LOD cloud: An empirical investigation. *Semantic Web*, 9(6):859–901, 2018.
[45] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *VLDB J.*, 22(5):665–687, 2013.
[46] J. A. Díaz and E. Fernández. A tabu search heuristic for the generalized assignment problem. *EJOR*, 132(1):22–38, 2001.
[47] D. C. do Nascimento, C. E. S. Pires, and D. G. Mestre. Exploiting block co-occurrence to control block sizes for entity resolution. *Knowl. Inf. Syst.*, 62(1):359–400, 2020.
[48] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, pages 85–96, 2005.
[49] X. L. Dong and D. Srivastava. *Big Data Integration*. Morgan & Claypool, 2015.
[50] C. F. Dorneles, R. Gonçalves, and R. dos Santos Mello. Approximate data instance matching: a survey. *KAIS*, 27(1):1–21, Apr 2011.
[51] U. Draisbach and F. Naumann. Dude: The duplicate detection toolkit. In *QDB*, 2010.
[52] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11):1454–1467, 2018.
[53] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *PVLDB*, 13(3), 2019.
[54] V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, and V. Christophides. Matching web tables with knowledge base entities: From entity lookups to entity embeddings. In *ISWC*, pages 260–277, 2017.
[55] V. Efthymiou, G. Papadakis, G. Papastefanatos, K. Stefanidis, and T. Palpanas. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Inf. Syst.*, 65:137–157, 2017.
[56] V. Efthymiou, G. Papadakis, K. Stefanidis, and V. Christophides. MinoanER: Schema-agnostic, non-iterative, massively parallel resolution of web entities. In *EDBT*, pages 373–384, 2019.
[57] V. Efthymiou, K. Stefanidis, and V. Christophides. Big data entity resolution: From highly to somehow similar entity descriptions in the web. In *IEEE Big Data*, pages 401–410, 2015.
[58] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.
[59] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.
[60] J. Esquivel, D. Albakour, M. Martinez-Alvarez, D. Corney, and S. Moussa. On the long-tail entities in news. In *ECIR*, 2017.
[61] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma. Dynamic constraints for record matching. *VLDB J.*, 20(4):495–520, 2011.
[62] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2(1):407–418, 2009.
[63] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.
[64] D. Firmani, B. Saha, and D. Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5):384–395, 2016.
[65] J. Fisher, P. Christen, Q. Wang, and E. Rahm. A clustering-based framework to control block sizes for entity resolution. In *SIGKDD*, pages 279–288, 2015.
[66] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4):385–408, 2003.
[67] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.
[68] C. Fu, X. Han, L. Sun, B. Chen, W. Zhang, S. Wu, and H. Kong. End-to-end multi-perspective matching for entity resolution. In *IJCAI*, pages 4961–4967, 2019.
[69] A. Gal. Tutorial: Uncertain entity resolution. *PVLDB*, 7(13):1711–1712, 2014.
[70] S. Galhotra, D. Firmani, B. Saha, and D. Srivastava. Robust entity resolution using random graphs. In *SIGMOD*, pages 3–18, 2018.
[71] N. Gao, S.-J. Huang, Y. Yan, and S. Chen. Cross modal similarity learning with active queries. *Pattern Recogn.*, 75(C):214–222, 2018.

[72] L. Getoor and A. Machanavajjhala. Entity resolution: Theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.
[73] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *TKDD*, 1(1):4, 2007.
[74] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, pages 601–612, 2014.
[75] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. Tan. Data integration: After the teenage years. In *PODS*, pages 101–106, 2017.
[76] Y. Govind, E. Paulson, P. Nagarajan, P. S. G. C., A. Doan, Y. Park, G. Fung, D. Conathan, M. Carter, and M. Sun. Cloudmatcher: A hands-off cloud/crowd service for entity matching. *PVLDB*, 11(12):2042–2045, 2018.
[77] A. Gruenheid, X. L. Dong, and D. Srivastava. Incremental record linkage. *PVLDB*, 7(9):697–708, May 2014.
[78] M. A. Gulzar, M. Interlandi, S. Yoo, S. D. Tetali, T. Condie, T. Millstein, and M. Kim. Bigdebug: Debugging primitives for interactive big data processing in spark. In *ICSE*, pages 784–795, 2016.
[79] S. Hajian, F. Bonchi, and C. Castillo. Algorithmic bias: From discrimination discovery to fairness-aware data mining. In *KDD*, 2016.
[80] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.
[81] O. Hassanzadeh and R. J. Miller. Creating probabilistic databases from duplicated data. *VLDB J.*, 18(5):1141–1166, 2009.
[82] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web. In *WebDB*, pages 129–134, 2000.
[83] M. A. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky. HIL: a high-level scripting language for entity integration. In *EDBT*, pages 549–560, 2013.
[84] M. A. Hernàndez and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.
[85] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.
[86] J. Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006.
[87] I. F. Ilyas and X. Chu. *Data Cleaning*. ACM, 2019.
[88] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. *PVLDB*, 3(1):429–438, 2010.
[89] E. Ioannou, C. Niederée, and W. Nejdl. Probabilistic entity linkage for heterogeneous information spaces. In *CAiSE*, 2008.
[90] E. Ioannou, N. Rassadko, and Y. Velegrakis. On generating benchmark data for entity matching. *J. Data Semantics*, 2(1):37–56, 2013.
[91] R. Isele and C. Bizer. Learning expressive linkage rules using genetic programming. *PVLDB*, 5(11):1638–1649, 2012.
[92] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.
[93] P. Jurczyk, J. J. Lu, L. Xiong, J. D. Cragan, and A. Correa. Fine-grained record integration and linkage tool. *BDR*, 82(11), 2008.
[94] A. Jurek, J. Hong, Y. Chi, and W. Liu. A novel ensemble learning approach to unsupervised record linkage. *Inf. Syst.*, 71:40–54, 2017.
[95] A. Karakasidis and E. Pitoura. Identifying bias in name matching tasks. In *EDBT*, pages 626–629, 2019.
[96] D. Karapiperis, A. Gkoulalas-Divanis, and V. S. Verykios. Summarization algorithms for record linkage. In *EDBT*, pages 73–84, 2018.
[97] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa. Low-resource deep entity resolution with transfer and active learning. In *ACL*, pages 5851–5861, 2019.
[98] X. Ke, M. Teo, A. Khan, and V. K. Yalavarthi. A demonstration of PERC: probabilistic entity resolution with crowd errors. *PVLDB*, 11(12):1922–1925, 2018.
[99] M. Kejriwal and D. P. Miranker. An unsupervised algorithm for learning blocking schemes. In *ICDM*, pages 340–349, 2013.
[100] M. Kejriwal and D. P. Miranker. A two-step blocking scheme learner for scalable link discovery. In *OM*, pages 49–60, 2014.
[101] M. Kejriwal and D. P. Miranker. A DNF blocking scheme learner for heterogeneous datasets. *CoRR*, abs/1501.01694, 2015.
[102] M. Kejriwal and D. P. Miranker. An unsupervised instance matcher for schema-free RDF data. *J. Web Sem.*, 35:102–123, 2015.
[103] A. R. Khan and H. Garcia-Molina. Attribute-based crowd entity resolution. In *CIKM*, pages 549–558, 2016.
[104] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12), 2016.
[105] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210, 2010.
[106] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1), 2010.
[107] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: Similarity measures and algorithms. In *SIGMOD*, pages 802–803, 2006.
[108] J. M. Kurtzberg. On approximation methods for the assignment problem. *J. ACM*, 9(4):419–439, 1962.
[109] S. Kushagra, H. Saxena, I. F. Ilyas, and S. Ben-David. A semi-supervised framework of clustering selection for de-duplication. In *ICDE*, pages 208–219, 2019.
[110] S. Kwashie, J. Liu, J. Li, L. Liu, M. Stumptner, and L. Yang. Certus: An effective entity resolution approach with graph differential dependencies (gdds). *PVLDB*, 12(6):653–666, 2019.
[111] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, and Z. Ghahramani. Sigma: simple greedy matching for aligning large knowledge bases. In *SIGKDD*, pages 572–580, 2013.
[112] F. Li, X. L. Dong, A. Langen, and Y. Li. Knowledge verification for longtail verticals. *PVLDB*, 10(11), 2017.
[113] G. Li, Y. Zheng, J. Fan, J. Wang, and R. Cheng. Crowdsourced data management: Overview and challenges. In *SIGMOD*, 2017.
[114] J. Li, J. Tang, Y. Li, and Q. Luo. Rimom: A dynamic multistrategy ontology alignment framework. *TKDE*, 21(8):1218–1232, 2009.
[115] D. Logothetis, S. De, and K. Yocum. Scalable lineage capture for debugging disc analytics. In *SoCC*, pages 17:1–17:15, 2013.
[116] Y. Ma and T. Tran. Typimatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In *WSDM*, pages 325–334, 2013.
[117] C. Mathieu, O. Sankur, and W. Schudy. Online correlation clustering. In *STACS*, pages 573–584, 2010.
[118] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *SIGKDD*, pages 169–178, 2000.
[119] W. McNeill, H. Kardes, and A. Borthwick. Dynamic record blocking: efficient linking of massive databases in mapreduce. In *QDB*, 2012.
[120] D. G. McVitie and L. B. Wilson. Stable marriage assignment for unequal sets. *BIT Numerical Mathematics*, 10(3), 1970.
[121] G. Mesnil, X. He, L. Deng, and Y. Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *INTERSPEECH*, pages 3771–3775, 2013.
[122] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *SIGMOD*, pages 19–34, 2018.
[123] C. Nanayakkara, P. Christen, and T. Ranbaduge. Robust temporal graph clustering for group record linkage. In *PAKDD*, 2019.
[124] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Morgan & Claypool, 2010.

[125] E. Nelson and J. Talburt. Entity resolution for longitudinal studies in education using oyster. In *IKE*, 2011.
[126] M. Nentwig, A. Groß, and E. Rahm. Holistic entity clustering for linked data. In *IEEE ICDM Workshops*, pages 194–201, 2016.
[127] M. Nentwig, M. Hartung, A. N. Ngomo, and E. Rahm. A survey of current link discovery frameworks. *Sem. Web*, 8(3):419–436, 2017.
[128] A. N. Ngomo and S. Auer. LIMES - A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*, 2011.
[129] M. Nickel and D. Kiela. Poincaré embeddings for learning hierarchical representations. In *NIPS*, 2017.
[130] A. Nikolov, V. S. Uren, E. Motta, and A. N. D. Roeck. Integration of semantically annotated data by the knofuss architecture. In *EKAW*, pages 265–274, 2008.
[131] J. Nin, V. Muntés-Mulero, N. Martínez-Bazan, and J. Larriba-Pey. On the use of semantic blocking techniques for data cleansing and integration. In *IDEAS*, pages 190–198, 2007.
[132] K. O'Hare, A. Jurek-Loughrey, and C. de Campos. A review of unsupervised and semi-supervised blocking methods for record linkage. In *Linking and Mining Heterogeneous and Multi-view Data*, pages 79–105. Springer, 2019.
[133] G. Papadakis, G. Alexiou, G. Papastefanatos, and G. Koutrika. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *PVLDB*, 9(4):312–323, 2015.
[134] G. Papadakis, K. Bereta, T. Palpanas, and M. Koubarakis. Multi-core meta-blocking for big linked data. In *SEMANTICS*, 2017.
[135] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In *WSDM*, pages 53–62, 2012.
[136] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *TKDE*, 25(12), 2013.
[137] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. Meta-blocking: Taking entity resolution to the next level. *TKDE*, 26(8), 2014.
[138] G. Papadakis, G. Papastefanatos, and G. Koutrika. Supervised meta-blocking. *PVLDB*, 7(14):1929–1940, 2014.
[139] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *EDBT*, pages 221–232, 2016.
[140] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas. A survey of blocking and filtering techniques for entity resolution. *CSUR*, 53(2).
[141] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB*, 9(9):684–695, 2016.
[142] G. Papadakis, L. Tsekouras, E. Thanos, N. Pittaras, G. Simonini, D. Skoutas, P. Isaris, G. Giannakopoulos, T. Palpanas, and M. Koubarakis. Jedai^3 : beyond batch, blocking-based entity resolution. In *EDBT*, pages 603–606, 2020.
[143] T. Papenbrock, A. Heise, and F. Naumann. Progressive duplicate detection. *TKDE*, 27(5):1316–1329, 2015.
[144] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.
[145] B. Ramadan and P. Christen. Forest-based dynamic sorted neighborhood indexing for real-time entity resolution. In *CIKM*, 2014.
[146] B. Ramadan, P. Christen, H. Liang, and R. W. Gayler. Dynamic sorted neighborhood indexing for real-time entity resolution. *J. Data and Information Quality*, 6(4):15:1–15:29, 2015.
[147] B. Ramadan, P. Christen, H. Liang, R. W. Gayler, and D. Hawking. Dynamic similarity-aware inverted indexing for real-time entity resolution. In *PAKDD Workshops*, pages 47–58, 2013.
[148] V. Rastogi, N. N. Dalvi, and M. N. Garofalakis. Large-scale collective entity matching. *PVLDB*, 4(4):208–218, 2011.
[149] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, 2017.
[150] O. F. Reyes-Galaviz, W. Pedrycz, Z. He, and N. J. Pizzi. A supervised gradient-based learning algorithm for optimized entity resolution. *Data Knowl. Eng.*, 112:106–129, 2017.
[151] S. V. Rice. Braided avl trees for efficient event sets and ranked sets in the simscript iii simulation programming language. In *WMC*, 2007.
[152] S. Rong, X. Niu, E. W. Xiang, H. Wang, Q. Yang, and Y. Yu. A machine learning approach for instance matching based on similarity metrics. In *ISWC*, pages 460–475, 2012.
[153] A. Saeedi, M. Nentwig, E. Peukert, and E. Rahm. Scalable matching and clustering of entities with FAMER. *CSIMQ*, 16:61–83, 2018.
[154] A. Saeedi, E. Peukert, and E. Rahm. Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In *ADBIS*, pages 278–293, 2017.
[155] A. Saeedi, E. Peukert, and E. Rahm. Using link features for entity clustering in knowledge graphs. In *ESWC*, pages 576–592, 2018.
[156] M. Sariyar, A. Borg, and K. Pommerening. Controlling false match rates in record linkage using extreme value theory. *Journal of biomedical informatics*, 44(4):648–654, 2011.
[157] A. D. Sarma, A. Jain, A. Machanavajjhala, and P. Bohannon. An automatic blocking mechanism for large-scale de-duplication tasks. In *CIKM*, pages 1055–1064, 2012.
[158] A. T. Schneider, A. Mukherjee, and E. C. Dragut. Leveraging social media signals for record linkage. In *WWW*, pages 1195–1204, 2018.
[159] C. Shao, L. Hu, J. Li, Z. Wang, T. L. Chung, and J. Xia. Rimom-im: A novel iterative framework for instance matching. *J. Comput. Sci. Technol.*, 31(1):185–197, 2016.
[160] L. Shu, A. Chen, M. Xiong, and W. Meng. Efficient spectral neighborhood blocking for entity resolution. In *ICDE*, 2011.
[161] G. Simonini, S. Bergamaschi, and H. V. Jagadish. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB*, 9(12):1173–1184, 2016.
[162] G. Simonini, L. Gagliardelli, S. Bergamaschi, and H. V. Jagadish. Scaling entity resolution: A loosely schema-aware approach. *Inf. Syst.*, 83:145–165, 2019.
[163] G. Simonini, G. Papadakis, T. Palpanas, and S. Bergamaschi. Schema-agnostic progressive entity resolution. *TKDE*, 31(6), 2019.
[164] Y. Sismanis, L. Wang, A. Fuxman, P. J. Haas, and B. Reinwald. Resolution-Aware Query Answering for Business Intelligence. In *ICDE*, pages 976–987, 2009.
[165] D. Song and J. Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *ISWC'11*.
[166] K. Stefanidis, V. Efthymiou, M. Herschel, and V. Christophides. Entity resolution in the web of data. In *WWW*, pages 203–204, 2014.
[167] R. C. Steorts, S. L. Ventura, M. Sadinle, and S. E. Fienberg. A comparison of blocking methods for record linkage. In *PSD*, 2014.
[168] W. Su, J. Wang, and F. H. Lochovsky. Record matching over query results from multiple web databases. *TKDE*, 22(4), 2010.
[169] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3), 2011.
[170] Z. Sun, W. Hu, and C. Li. Cross-lingual entity alignment via joint attribute-preserving embedding. In *ISWC*, 2017.
[171] Z. Sun, W. Hu, Q. Zhang, and Y. Qu. Bootstrapping entity alignment with knowledge graph embedding. In *IJCAI*, 2018.
[172] Z. Sun, Q. Zhang, W. Hu, C. Wang, M. Chen, F. Akrami, and C. Li. A benchmarking study of embedding-based entity alignment for knowledge graphs. *CoRR*, abs/2003.07743, 2020.

[173] S. Thirumuruganathan, S. A. P. Parambath, M. Ouzzani, N. Tang, and S. Joty. Reuse and adaptation for entity resolution through transfer learning. *CoRR*, abs/1809.11084, 2018.

[174] B. D. Trisedya, J. Qi, and R. Zhang. Entity alignment between knowledge graphs using attribute embeddings. In *AAAI*, 2019.

[175] S. M. Van Dongen. *Graph clustering by flow simulation.* PhD thesis, Utrecht University, 2000.

[176] M. van Erp, P. N. Mendes, H. Paulheim, F. Ilievski, J. Plu, G. Rizzo, and J. Waitelonis. Evaluating entity linking: An analysis of current benchmark datasets and a roadmap for doing a better job. In *LREC*, 2016.

[177] V. Verroios and H. Garcia-Molina. Entity resolution with crowd errors. In *ICDE*, pages 219–230, 2015.

[178] V. Verroios, H. Garcia-Molina, and Y. Papakonstantinou. Waldo: An adaptive human interface for crowd entity resolution. In *SIGMOD*, pages 1133–1148, 2017.

[179] N. Vesdapunt, K. Bellare, and N. N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.

[180] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk - A link discovery framework for the web of data. In *LDOW*, 2009.

[181] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[182] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, pages 469–480, 2014.

[183] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.

[184] J. Wang, G. Li, J. X. Yu, and J. Feng. Entity matching: How similar is similar. *PVLDB*, 4(10):622–633, 2011.

[185] S. Wang, X. Xiao, and C. Lee. Crowd-based deduplication: An adaptive approach. In *SIGMOD*, pages 1263–1277, 2015.

[186] X. Wang, L. M. Haas, and A. Meliou. Explaining data integration. *IEEE Data Eng. Bull.*, 41(2):47–58, 2018.

[187] Y. Wang, S. Song, L. Chen, J. X. Yu, and H. Cheng. Discovering conditional matching rules. *TKDD*, 11(4):46:1–46:38, 2017.

[188] Z. Wang, Q. Lv, X. Lan, and Y. Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. In *EMNLP*, 2018.

[189] M. Weis and F. Naumann. Detecting duplicate objects in XML documents. In *IQIS*, pages 10–19, 2004.

[190] M. Weis and F. Naumann. Detecting duplicates in complex XML data. In *ICDE*, page 109, 2006.

[191] M. J. Welch, A. Sane, and C. Drome. Fast and accurate incremental entity resolution relative to an entity knowledge base. In *CIKM*, pages 2667–2670, 2012.

[192] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[193] S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *TKDE*, 25(5):1111–1124, 2013.

[194] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity Resolution with Iterative Blocking. In *SIGMOD*, pages 219–232, 2009.

[195] D. T. Wijaya and S. Bressan. Ricochet: A family of unconstrained algorithms for graph clustering. In *DASFAA*, pages 153–167, 2009.

[196] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[197] V. K. Yalavarthi, X. Ke, and A. Khan. Select your questions wisely: For entity resolution with crowd errors. In *CIKM*, 2017.

[198] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search - The Metric Space Approach.* Kluwer, 2006.

[199] C. J. Zhang, R. Meng, L. Chen, and F. Zhu. Crowdlink: An error-tolerant model for linking complex records.

[200] F. Zhang, Z. Gao, and K. Niu. A pruning algorithm for meta-blocking based on cumulative weight. In *JPCS*, volume 887, 2017.

[201] Q. Zhang, Z. Sun, W. Hu, M. Chen, L. Guo, and Y. Qu. Multi-view knowledge graph embedding for entity alignment. In *IJCAI*, 2019.

[202] W. Zhang, H. Wei, B. Sisman, X. L. Dong, C. Faloutsos, and D. Page. Autoblock: A hands-off blocking framework for entity matching. In *WSDM*, pages 744–752. ACM, 2020.

[203] C. Zhao and Y. He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *WWW*, pages 2413–2424, 2019.

[204] Q. Zheng, X. Diao, J. Cao, X. Zhou, Y. Liu, and H. Li. Multi-modal space structure: a new kind of latent correlation for multi-modal entity resolution. *CoRR*, abs/1804.08010, 2018.

[205] H. Zhu, R. Xie, Z. Liu, and M. Sun. Iterative entity alignment via joint knowledge embeddings. In *IJCAI*, 2017.