data

MDPI

Article

Detective Gadget: Generic Iterative Entity Resolution over Dirty Data

Marcello Buoncristiano [1,†], Giansalvatore Mecca [2,†], Donatello Santoro [2,†] and Enzo Veltri [2,*,†]

[1] Svelto!—Big Data-Cleaning and Analytics, 85100 Potenza, Italy; marcello.buoncristiano@svelto.tech
[2] Dipartimento di Ingegneria, Università degli Studi della Basilicata, 85100 Potenza, Italy; giansalvatore.mecca@unibas.it (G.M.); donatello.santoro@unibas.it (D.S.)
[*] Correspondence: enzo.veltri@unibas.it
[†] These authors contributed equally to this work.

Abstract: In the era of Big Data, entity resolution (ER), i.e., the process of identifying which records refer to the same entity in the real world, plays a critical role in data-integration tasks, especially in mission-critical applications where accuracy is mandatory, since we want to avoid integrating different entities or missing matches. However, existing approaches struggle with the challenges posed by rapidly changing data and the presence of dirtiness, which requires an iterative refinement during the time. We present Detective Gadget, a novel system for iterative ER that seamlessly integrates data-cleaning into the ER workflow. Detective Gadgetemploys an alias-based hashing mechanism for fast and scalable matching, check functions to detect and correct mismatches, and a human-in-the-loop framework to refine results through expert feedback. The system iteratively improves data quality and matching accuracy by leveraging evidence from both automated and manual decisions. Extensive experiments across diverse real-world scenarios demonstrate its effectiveness, achieving high accuracy and efficiency while adapting to evolving datasets.

Keywords: entity resolution; iterative; algorithms; design; performance

check for updates

Citation: Buoncristiano, M.; Mecca, G.; Santoro, D.; Veltri, E. Detective Gadget: Generic Iterative Entity Resolution over Dirty Data. *Data* **2024**, *9*, 139. https://doi.org/10.3390/data9120139

Academic Editor: Florentino Fdez-Riverola

Received: 30 October 2024
Revised: 15 November 2024
Accepted: 19 November 2024
Published: 25 November 2024

1. Introduction

Entity Resolution is the process of identifying which of the records in a data collection refer to the same entity of the real world, a problem also known under the name of deduplication, record linkage, or merge–purge.

It is considered a significant research problem with data-cleaning, integration, and mining applications. In recent years, it has assumed a key role in *information-extraction applications* [1], which collect and integrate descriptions of objects of the real world taken from heterogeneous data sources, typically on the Web.

It can be seen how entity resolution (ER in the following) is a key technical problem in this process. Given its importance, a wealth of research has been devoted to the problem [2–4]. Previous evaluations (e.g., [5]) have shown that fully automatic approaches often struggle to reach the level of accuracy required in *mission-critical applications* and that running times are typically quite high even for tasks of a few thousand tuples. Mission-critical applications are the ones where high-quality results of the ER process are required.

This paper stems from several experiences conducted by our group with mission-critical ER in various domains, including research papers and publication venues (journals, conferences, and publishers), events, medical records [6,7], and personal data, for public and private organizations. In these experiences, we have made a few observations that stand at the core of our approach, as follows.

a *Rapid changes*: Web data sources and data lakes used in information-extraction applications typically change very rapidly [8,9], so the ER process needs to be repeated frequently to obtain fresh information about the real-world entity they represent.

b *Dirty data*: a critical problem is that data at the sources are often dirty [10]. Systems in the literature [3] typically assume that the input records undergo a preliminary data-cleaning phase before being fed to the ER algorithm. However, we will show in the paper that it is often impossible to clearly separate the data-cleaning phase from the ER phase since most of the dirtiness in the data is discovered only *after* an initial entity resolution has been produced. For this reason, data architects may need to iterate data-cleaning, and ER runs until an output of acceptable quality is produced.

c *Accuracy Requirements and Cost*: ER is notoriously expensive because it may require running several comparisons that are quadratic in the size of the input records. In addition, as we mentioned above, automatic approaches often give poor performance [5]. Experience says that real-life organizations that use ER results for mission-critical tasks often require accuracy higher than 95% and, therefore, hardly rely solely on automatic matches based on similarity functions. For example, in research evaluation, it is crucial to accurately match conference ratings from different sources to assess a conference's scientific value. False positives may severely impact the quality of evaluations and, therefore, cannot be tolerated. False negatives also have a negative impact on quality. Similarly, in developing integrated systems for e-healthcare, it is not acceptable to match medical records for different patients. This is why it is usually necessary to involve humans in the loop [11,12], either by relying on small groups of experts or on crowdsourcing to perform clerical reviews. However, human labor is expensive and can considerably slow down the process. Consequently, it is mandatory to design frameworks that can integrate automatic processing and human inputs as effectively as possible.

Based on these observations, it is easy to see that the ER process needs to be iterative. Also, the first iteration is usually quite expensive, both in terms of execution times and money, especially in mission-critical scenarios where humans need to be involved. Therefore, iterating the process is acceptable only as long as successive iterations can be computed with minimal additional costs.

Suppose we intend to evaluate the reputation of computer-science conferences, i.e., obtain the ranking or class and obtain the H-index. To do this, we may integrate some of the various conference rankings available on the Web, like the Australian CORE ranking [1], which assigns a class of merit to each conference, and the LiveSHINE ranking, which estimates H-indexes for conferences based on Google Scholar data. After the integration and the merge of conferences, we have a global reputation for such conferences. In this ER scenario we might have a DBLP [2] recordset with attributes confName, acronym, URL, a CORE recordset, with attributes confName, acronym, class, and one more from LiveSHINE with attributes confName, acronym, H-index. One example is shown in Figure 1.

Throughout the paper, we shall refer to this example. We name it the *conference-ranking example*.

DBLP	**id**	**confName**	**acronym**	**URL**
	r_1:	ACM SIGMOD Conf. on Management of Data	ACM SIGMOD	http://dblp.uni-trier.de/sigmod
	r_2:	Int. Conf. on Very Large Databases	VLDB	http://dblp.uni-trier.de/vldb
	r_3:	Int. Conf. on Management of Data	COMAD	http://dblp.uni-trier.de/comad
CORE	**id**	**confName**	**acronym**	**class**
	r_4:	ACM Int. Conf. on Management of Data	SIGMOD	A+
	r_5:	Very Large Databases	VLDB	A+
	r_6:	Int. Conf. on Management of Data	COMAD	B
SHINE	**id**	**confName**	**acronym**	**H-index**
	r_7:	Int. Conf. on Management of Data	SIGMOD	143
	r_8:	Int. Conf. on Very Large Databases	VLDB	134

Figure 1. Sample records for the conference-ranking example.

In the example, multiple sources provide information about computer-science conferences. These sources have inconsistencies: (1) The DBLP dataset lists conferences with their names, acronyms, and URLs. (2) The CORE dataset provides rankings but uses slightly different names or acronyms. (3) The LiveSHINE dataset includes H-indices for conferences but may use ambiguous names like “Int. Conf. on Management of Data”, which could refer to either ACM SIGMOD or COMAD.

The goal is to integrate these datasets to correctly match records that refer to the same conference, ensuring that ACM SIGMOD and COMAD are not mistakenly grouped while correctly merging all VLDB entries.

In this paper, we study this important problem, called *iterative entity resolution in the presence of dirty data*: we are given an ER task, and we assume that an initial ER has been derived (using any of the techniques in [2–4]), possibly collecting human decisions during the process; then, the input records are updated (either at the sources, or as a result to some data-cleaning activity), and we need to iterate the ER step until no further updates are needed.

Within such an interactive ER process, we assume that any of the techniques available in the literature has been used to compute a first approximation of the ER, which we take as input. Then, our focus is on designing a systematic set of steps for computing successive iterations in such a way as to obtain the best trade-off between quality and costs (e.g., the number of manually labeled examples and computational costs).

To do this, we develop fast algorithms and data structures in order to reuse automatic and human decisions about matches so that the cost of running successive ER steps drops significantly. Previous approaches have materialized match decisions to avoid recomputing them. In the paper, we show that this is extremely slow in many cases and makes frequent iterations completely unfeasible.

In light of this, we develop a novel entity-resolution system called Detective Gadget.

a Detective Gadget adopts a *generic* approach to entity resolution, i.e., it may incorporate a variety of match functions in order to establish if two records match each other, seen largely as black boxes; in this respect, Detective Gadget allows for the maximum flexibility in incorporating known techniques to speed up the initial ER step; for example, it may easily incorporate inputs provided by human experts or the crowd along with automatic match functions, or machine-learning models, in such a way that users may fine-tune the trade-off between the costs and quality of the ER.

b Differently from previous approaches, Detective Gadget does not assume that a data-cleaning step has been performed prior to the entity-resolution phase; on the contrary, it handles data-cleaning and entity resolution in an integrated fashion, using a greedy-match algorithm that uses both positive and negative evidence about the matches to refine the entity resolution while at the same time cleaning up the original datasets;

c At the core of the system stands a very fast match algorithm capable of leveraging past positive and negative evidence in an extremely efficient way. The algorithm is based on a novel technique called *alias-based hashing*, which relies on shadow values—i.e., additional values used by the algorithm in addition to the original ones—for the input records, called *aliases*. Intuitively, aliases are alternative versions of the value of an attribute, which may be updated to make two different but matching values or records identical to each other.

d This aggressive hash-based matching technique allows for the development of a novel hash-based algorithm to recompute groups of matching records that make the execution of successive iterations orders of magnitude faster than previous approaches. In fact, as will be discussed in the following, we introduce a novel blocking technique to limit the number of pairwise comparisons needed to identify similar values, called *inverted blocking* that usually significantly improves performance in highly duplicated datasets. The interaction between hashing and inverted blocking represents a novel part of our contribution.

Detective Gadget makes several novel contributions to the solution of the iterative ER problem:

i We develop an iterative fixpoint ER algorithm that can handle data-cleaning and data-deduplication decisions in an integrated fashion. The algorithm uses greedy matching to quickly generate an initial ER and then runs check functions to detect potential inconsistencies and suggest repair instructions to clean dirty values. After the original values have been fixed, the process is iterated until no new matches and errors are detected.

ii At the core of the ER algorithm stands a two-step match algorithm, which clearly separates the search for equality-based matches (using a doubly-linked hash data structure) from the discovery of value similarities. The separation of the two phases gives opportunities for a number of optimizations. To do this, in analogy with keys in relational databases, we develop a theory of *identifiers* for ER purposes; we propose a taxonomy of identifiers and notice that, due to dirtiness in the original data sources, identifiers are often *weak*, in the sense that they do not always provide definitive evidence about matching records. Stronger IDs, however, may improve the efficiency of the process.

iii The match-algorithm core component is the use of equivalence classes and value aliases to restore previous knowledge about matches and mismatches so that the process can be iterated several times with very little cost; these techniques are designed to accommodate human inputs efficiently. This reflects our vision that the three crucial activities in ER—cleaning the data, matching them, and performing clerical reviews—cannot be handled as separate activities and need to be integrated within a single process.

iv We introduce a new iterative workflow that enables the integration of human decisions into the ER process. The iterative nature allows the evaluation of the quality of the ER process and, if needed, to reiterate to further improve the quality of the results.

v We conduct an ample experimental evaluation and compare our algorithms to several of the systems in the literature on different ER tasks. We show how Detective Gadget represents an effective solution to the problem of handling dirty recordsets that are frequently updated in mission-critical scenarios.

The paper is organized as follows: the following section introduces several preliminary notions (Section 2.1) and then provides an overview of entity-resolution concepts (Section 2.2). We use them to introduce the main challenges of the process and further elaborate on the contributions made by this paper. We define the data model and the definition of the iterative ER (Section 2.3), and we present an algorithm to solve the iterative ER (Section 2.7). Section 3 presents the experimental results and the discussion of the results. Finally, Sections 4 and 5 conclude the paper with related works and conclusions.

2. Materials and Methods

2.1. Preliminaries

The input to an entity-resolution process is a set of structured record collections, also called *recordsets*. Each collection contains structured records, i.e., sets of attribute-value pairs.

As will be detailed in our experimental evaluation (Section 3), we considered several ER scenarios, both from real-life recordsets and known benchmarks.

Consider again the *conference-ranking example*. As is apparent, recordsets may have different schemas. In addition, they usually lack clear identifiers. In our example, an acronym might seem a good identifier, but it fails to match ACM SIGMOD and SIGMOD. The same applies to confName. Recordsets also have widely different levels of quality. It is typically assumed [3] that the original records have undergone a data-cleaning phase before entering the ER process, in order to standardize and repair their values in such a way that the inputs to the actual ER phase are error-free. In this paper, we do not make this assumption. To simplify the treatment, and without loss of generality, we will only assume that attribute labels have been normalized in such a way that attributes with the same name have the same semantics in all sources and no ambiguous attribute names are presented in the datasets [13,14].

Generic Entity Resolution

To discuss the ER process, we follow the nice abstraction introduced in the SWOOSH framework [15] under the name of *generic ER*. After the initial exploration of the data [16] and data-cleaning step, which is considered a separate phase with respect to the actual ER phase, the basic technique consists of comparing records in order to identify *matches*. Matching records are then merged to obtain a representation of the entity to which they refer. Matching and merging records are, therefore, the two main operations of ER. Generic ER makes limited assumptions about the actual match and merge functions, which are considered black boxes in order to be able to incorporate approaches of different costs and quality.

Our main focus in this paper is on the match phase, which is by far the most delicate and expensive. Speaking of merging, there are different classes of merge functions [17], but it has been shown [15] that those based on the union operation are "well behaved", in the sense that they preserve all the information in the original records, and can be used subsequently for further elaborations. Therefore, in the following we shall assume that the merge operation is simply the union of all records that match with one another. Using this simple form of merge, at the end of the process, each entity will be represented as a set of records, which we call a *group*. Consider the conference-ranking example, and assume the set of input records is the one in Figure 1.

Also assume we match two record conferences if they have equal acronyms or similar names. Then, the VLDB conference will be represented by the following group of records (attribute names are omitted for brevity):

$$\begin{array}{ll} \{ & r_2 = [\text{Int. Conf. on Very Large Databases, VLDB, http://...}] \\ & r_5 = [\text{Very Large Databases, VLDB, A+}] \\ & r_8 = [\text{Int. Conf. on Very Large Databases, VLDB, 134}] \quad \} \end{array}$$

With this in mind, we concentrate on the match operation. In its essence, a match function compares the values of attributes in a pair of records to measure their similarity and ultimately decide if they match. The plenitude of approaches [2–4] that have been proposed to match records in ER tasks can be classified along several lines.

We concentrate only on match algorithms that are based on *pairwise* record comparisons, as opposed to *clustering*-based algorithms, where groups of matching records are identified by partitioning the input records according to algorithms that take into consideration the global space of objects. On the contrary, pairwise-based approaches rely on local decisions, and are often preferred since they are easier to write and characterize, also for non-expert users. Moreover, pairwise-based approaches could be easily integrated into a crowdsourcing environment [11,18,19].

Of the many possible classifications of pairwise comparison techniques that are found in the literature—supervised, learning-based techniques vs. unsupervised techniques; probabilistic vs. non-probabilistic; confidence-based vs. exact; single-type vs relational [2–4]—we find it useful to divide match functions into two categories:

- *record-level functions* compare records by looking at values in their entirety to establish whether there is a match or not;
- *attribute-level functions* compare the value of attributes one at a time. In our conference-ranking example, they might be used to say that two conference records match if they have similar names or equal acronyms.

Computing matches is by far the most expensive step in the entity-resolution process. Any algorithm that relies on pairwise record comparisons needs, in the worst case, to process comparisons in the order of $O(n^2)$, where n is the total number of input records. Even with a few thousand records, this number raises to the millions. In addition, string-similarity comparisons may be slow, much slower than numeric comparisons. This explains why a large fraction of the research work on this subject has been devoted to optimizing the match phase, either by reducing the number of comparisons, for example, by dividing records into blocks and comparing records within blocks only, or by parallelizing the computation of similarities over a distributed cluster [2,3,20].

Regardless of the actual strategy that has been used to compute and optimize the comparisons, in the end, a typical ER match function would classify record pairs into three different groups:

- the first group contains pairs of records that are considered to be definitive matches; these are ready to be merged in order to build groups corresponding to different entities;
- the second group contains pairs of records that are considered to be nonmatching, i.e., they are too different from one another;
- the third group contains pairs of records that are possible matches; typically, they exhibit some level of similarity, but this is not sufficient to motivate a clear match; these record pairs require a *clerical-review* phase, i.e., they need to be inspected by a human expert that labels them as matches or non-matches.

2.2. Overview of the Approach

Based on the notions introduced above, we are ready to summarize the main ideas upon which we build our proposal.

2.2.1. Attribute Classification

Web data sources used in information-extraction applications typically change very rapidly, so the ER process needs to be repeated frequently to generate fresh results. In this respect, we find it useful to classify attributes into two categories.

We call *identifying attributes* (or simply *IDs*) the ones that are used by the functions to identify matching records. We call *descriptive attributes* all the others. In our conference-ranking example, it is natural to pick up as IDs the confName and acronym attributes; all other attributes—class and H-index—are descriptive.

We notice that IDs (like conference names, publication titles, and event descriptions) tend to be stable, while descriptive attributes (like H-indexes, citation numbers, check-ins, or likes) may vary over time. We want to design the ER process in such a way that it may be efficiently iterated to incorporate new data. We, therefore, need to design an iterative algorithm that can reuse past knowledge about matching IDs to quickly reconstruct record groups and analyze new values of descriptive attributes to obtain fresh information about the real-world entity they represent.

2.2.2. Iteratively Cleaning Dirty Data

The iterative nature of the process is made even more crucial if we think that data at the original sources usually have very different levels of quality, i.e., while there may be a few curated sources that provide data of high quality, we expect most data sources to provide dirty data.

An even more serious problem is that the kind of dirtiness we face often makes it impossible to clearly separate the data-cleaning phase from the ER phase. To see this, consider our conference example. Following the traditional two-phase ER process, we need to inspect the various sources before conducting any ER activity to study their quality and repair inconsistencies. There is a very rich literature in data-cleaning and data repairing [10,21,22]. Most of the techniques are based on using constraints defined over the sources to detect and repair inconsistencies [23,24] or involve humans in the loop to detect constraints and repair data [25].

These techniques are often of limited use when dealing with ER–oriented recordsets. There are several reasons for this, the main one being that most of the dirtiness in the data is discovered only *after* an initial ER has been produced. Consider our example above;

by looking at the three recordsets in Figure 1, all of them look clean. However, as soon as records are matched, we generate a group like the one below (recall that we consider conference titles as an identifying attribute):

$$\{ \; r_1 = \text{[ACM SIGMOD Conf. ..., ACM SIGMOD, http://...]}$$
$$r_3 = \text{[Int. Conf. on Management of Data, COMAD, http ...]}$$
$$r_4 = \text{[ACM Int. Conf. on Management ..., SIGMOD, A+]}$$
$$r_6 = \text{[Int. Conf. on Management of Data, COMAD, B]}$$
$$r_7 = \text{[Int. Conf. on Management of Data, SIGMOD, 143]} \quad \}$$

We are putting together records that refer to two completely different entities, namely the ACM SIGMOD Conference and the COMAD Conference on Data. This is due to the name used to refer to the SIGMOD conference in the LiveSHINE dataset (*Int. Conf. on Management of Data*). To clean the data, that name should be changed to something more specific to the ACM SIGMOD Conference, but we can discover this only after we have brought data together and analyzed the output.

In fact, we see these data-cleaning instructions as another facet of the process of generating new, updated records for the data sources at hand. In both cases—new, fresh records at the sources and records updated as a result of data-cleaning modifications—we need to repeat the ER process. Relying on a very fast match algorithm is mandatory in this respect.

2.2.3. Integrating Human Inputs

As we mentioned, experience tells us that real-life organizations that use ER results for mission-critical tasks hardly rely on matches based on fully automatic algorithms. Broadly speaking, the accuracy of an ER process is usually measured by comparing the set of matching record pairs it discovers with some gold standard established by a human expert. In mission-critical applications, it is customary to require precision and recall, for example, above 95%. Automatic algorithms are often quite far from this threshold [5].

This is why it is often necessary to involve humans in the loop, either by relying on small groups of experts or on crowdsourcing [12], to perform *clerical reviews*. However, human labor is expensive and can considerably slow down the process. Consequently, it is mandatory to design frameworks that can integrate automatic processing and human inputs as effectively as possible.

The three novel techniques that stand at the core of our approach are called *greedy matching*, *check functions and cell changes*, and *value aliases*. These are discussed in the following paragraphs.

2.2.4. Greedy Matching

It is common to think of ER as the process of matching records by searching for similarities among their attributes. However, experience tells us that most of the matches are based on *identical* values, not similar ones. To see this, consider the table in Figure 2, in which we summarize some statistics about a number of ER tasks we have studied in Section 3.

task	#records	#matches	#equality matches	#approx. matches	% approx. matches
1. Journals	92,756	59,801	59,283	518	<1%
2. Conferences	3667	2059	1904	155	7%
3. Publications	45,648	22,373	17,327	5046	22%
4. Events	9742	3327	3124	203	6%
5. CDs	9760	233	175	58	24%
6. Restaurants	864	112	108	4	4%

Figure 2. Statistics about matches in various ER tasks. See Section 3 for a detailed description.

We classify matches in two categories: (a) equality matches, i.e., matches that are discovered based on identical values on one or multiple attributes—for example, equal names or ISSN for journals, equal names or acronyms for conferences, equal titles, places, and dates for events, and so on; (b) approximate matches, i.e., matches that were discovered based on similar values on one or multiple attributes.

A striking observation is that in all cases, equality matches were by far the most frequent case. This, in turn, leads to another crucial observation: while computing similarities is typically slow since it has a quadratic cost, matches based on identical values can be found in nearly-linear time.

For example, given two sets of n_1, n_2 conferences, we typically need to run $O(n_1 \times n_2)$ string comparisons to find those with similar names. Blocking [3]—i.e., dividing the recordset into subsets in order to conduct the search for matches only within subsets—may indeed help to alleviate the problem, but at the same time, it may also impact the quality of the results, since it may prevent the ER system from comparing records in different blocks that correspond to the same entity. For this reason, blocking must be used with great care in mission-critical scenarios.

On the contrary, matches between identical attributes may be found much more efficiently using hashing. In fact, several ER algorithms [2,15] have used hash-based data structures during the computation of similarities to speed up the match process to some extent.

Detective Gadget pushes this idea to the extreme: differently from previous approaches that mix similarity computations and hashing, we completely separate the search for equality-based matches from the one for similarity-based ones. This gives us several advantages.

First, it eliminates the need to compare records for which a match has already been identified with each other. In fact, we consider record groups obtained during the hash-based matching phase as blocks and only compute similarities among records belonging to different blocks. We call this *inverted blocking*.

Second, it allows us to further prune the number of string comparisons by reasoning on the properties of the input collections. If, for example, we have additional knowledge about the properties of records of one of the input collections—for example, that records in a dataset R have *strong IDs*, i.e., curated IDs that do not require further cleaning—after the hash phase, we avoid the search for similarities among groups that contain records of R with different IDs, because all meaningful matches have already been identified.

2.2.5. Check Functions

This separation alone greatly improves running times.There is, however, a negative counterpart to this speed improvement. In fact, aggressively matching records based on value equalities may generate wrong groups as the one about the SIGMOD conference discussed in our example above. To handle this problem and gather precious information that will drive our iterative data-cleaning process, we not only rely on positive evidence about record matches, but we also analyze groups to gather *negative* evidence about mismatching records that have been put together by mistake.

In fact, along with matching functions, we introduce *check functions*, i.e., functions that are used to compare candidate matching records within one group and suggest which of them are very different from each other and, therefore, may represent mismatches. In our conference-ranking example, a check function may detect that there is a strong difference in ranks between record r_4 (the real SIGMOD record, ranked A+) and record r_6 (the COMAD record, ranked B) or also that acronyms (SIGMOD, COMAD) are significantly different from one another, and therefore suggest that as a mismatch.

Notice that the idea of using *negative rules* for ER [26] is not new. However, the assumption that data are dirty forces us to reconsider the way in which negative evidence is used regarding previous approaches; in fact, we want not only to break the incorrect group but also to repair the original data so that this problem does not arise in the future. Therefore, we assume that a check rule also suggests (possibly with the help of a human expert) a set of updates to the original values that remove the ambiguity.

2.2.6. Value Aliases

In essence, after a first run of the matching algorithm, we have accumulated positive knowledge about matching values, i.e., that the following values are matches:

m_1 : "ACM SIGMOD Conf. on Manag. of Data"
$\approx$ "ACM Int. Conf. on Management of Data"
m_2 : "ACM SIGMOD Conf. on Manag. of Data"
$\approx$ "Int. Conf. on Management of Data"
..

We typically also have some negative knowledge. This suggests *cell corrections* repair dirty values in order to avoid mismatches. For example:

ch_1 : r_3.confName = "Int. Conf. on Management of Data"
:= "COMAD Int. Conf. on Management of Data"
ch_2 : r_6.confName = "Int. Conf. on Management of Data"
:= "COMAD Int. Conf. on Management of Data"
..

Previous approaches [15] have materialized these match results in such a way as to speed up the computation of successive steps. We notice, however, that storing pairs of matching strings or record descriptions simply removes the cost of recomputing the string similarity but does not remove the quadratic cost. Assume, in fact, that we have stored the results of the initial $O(n_1 \times n_2)$ comparisons of our conference titles and want to reuse them during the second iteration. This will remove the need to compute those expensive string comparisons but will still take $O(n_1 \times n_2)$ steps to restore the matches.

We intend to push the idea of using hashing to discover matches to the extreme. In order to do this, we do the following:

i with each attribute value within the input record sets, we associate an *alias*; the alias is initially equal to the original value but can be changed later on for the purpose of speeding the match phase;

ii before each successive iteration, we analyze the set of matches that we have already discovered and use them to induce *equivalence classes* of values. In our example above, matches m_1, m_2 induce the following equivalence class of values:

"ACM SIGMOD Conf. on Manag. of Data"
"ACM Int. Conf. on Management of Data"
"Int. Conf. on Management of Data"
..

For each equivalence class $\mathcal{C}$, we pick up a representative $v_{\mathcal{C}}$ value (say, the first value in our example);

iii before restarting the match, we scan the original record sets, and whenever we find a value v that belongs to an equivalence class $\mathcal{C}$ among the ones identified at the step above, we set its value alias to the representative value $v_{\mathcal{C}}$. This step can be conducted in linear time using hashing. At the end of the process, the original values have remained intact, but the value aliases have changed as follows :

r_1.confName = "ACM SIGMOD Conf. on Manag. of Data"
alias : "ACM SIGMOD Conf. on Manag. of Data"
r_4.confName = "ACM Int. Conf. on Management of Data"
alias : "ACM SIGMOD Conf. on Manag. of Data"
r_7.confName = "Int. Conf. on Management of Data"
alias : "ACM SIGMOD Conf. on Manag. of Data"
..

iv Before doing that, we assume that values of the original datasets are repaired by changing their aliases as suggested by the negative rules, i.e.:

r_3..confName = "Int. Conf. on Management of Data"
alias : "COMAD Int. Conf. on Management of Data"
r_6..confName = "Int. Conf. on Management of Data"
alias : "COMAD Int. Conf. on Management of Data"
..

v We start the new run of the matching algorithm; in doing that, we hash based on aliases, not original values. It can be seen as those that were similarities have now become identities and can be restored in a much faster way during the hash phase without incurring the quadratic cost.

In fact, following these changes, hashing with respect to equal titles and equal aliases alone provides us with the right groups, and no more search for similarities is needed (aliases are omitted for readability):

{ r_1 = [ACM SIGMOD Conf. ..., ACM SIGMOD, http://...]
r_4 = [ACM Int. Conf. on Management of ..., SIGMOD, A+]
r_7 = [Int. Conf. on Management of Data, SIGMOD, 143] }
{ r_3 = [Int. Conf. on Management of Data, COMAD, http://...]
r_6 = [Int. Conf. on Management of Data, COMAD, B] }
{ r_2 = [Int. Conf. on Very Large Databases, VLDB, http://...]
r_5 = [Very Large Databases, VLDB, A+]
r_8 = [Int. Conf. on Very Large Databases, VLDB, 134] }

2.3. Data Model and Task Definition

We now introduce the data model we use in our setting. Our data model consists of heterogeneous sets of tuples that are essentially relational tables. However, in ER terminology, tuples are often called *records*, and tables *record sets*. In the following, we shall use the two terminologies interchangeably.

Each record set R is a set of records, i.e., sets of attribute-value pairs of the form $r = (A_0 : v_0, \ldots, A_k : v_k)$. We assume that each record has a globally unique *record ID*, r_{id}. We denote by $r_{id}.A_i$ the value of attribute A_i in record r_{id}, also called a *cell* in the record set. Records may contain null values, denoted by *null*.

With each cell $r_{id}.A_i$, we associate a *value alias*, denoted by $alias(r_{id}.A_i)$, i.e., a secondary cell that will be used during the ER to speed up the computation of matches. Initially, the value of the alias cell is the same as the one of the main cell, but it may be changed at a later time. It is important to note that, in the following treatment, unless otherwise specified, whenever we refer to the value of record r for attribute A_i we shall actually refer to $alias(r.A_i)$, i.e., to the value of the alias cell. This allows us to leave the original instance unmodified while updating alias cells in order to carry on the ER process.

Recall that we assume that attribute names within record sets have been normalized, i.e., attributes with the same semantics have the same name in all record sets. It is fairly straightforward to extend the framework by introducing explicit attribute mappings to remove this assumption; we prefer to avoid this since it makes the technical development more involved without really improving the generality of the framework.

We call an *instance*, I, a collection of record sets. We assume that an instance I could contain multiple records that refer to the same entity.

2.3.1. Tuple Groups

In its most basic formulation, an ER task is defined as follows: we are given as inputs n record sets $R_1, R_2, \ldots R_n$, $n > 0$, and a set of *matching functions* $f_1, f_2, \ldots, f_m$, $m > 0$. The goal is to identify which of the input records represent the same entities of the real world. We denote the fact that records r_1, r_2 *semantically represent* or *refer* to the same entity by the symbol $r_1 \doteq r_2$. More formally, the output of an ER task is a set of *tuple groups* $\mathcal{G} = \{g_1 \dots g_k\}$, $k > 0$, where each $g_i \in \mathcal{G}$ is a non-empty set of tuples of $R_1, R_2, \dots R_n$ such that:

- each tuple t_i in any of the input record sets, $R_1, R_2, \dots R_n$, occurs in exactly one group $g_j \in \mathcal{G}$;
- all tuples in a group $g \in \mathcal{G}$ semantically represent the same entity, i.e., for each $t_i, t_j \in g$, it is the case that $t_i \doteq t_j$;
- tuples in different groups semantically represent different entities, i.e., for each $t_i \in g_i, t_j \in g_j$, if $i \neq j$ then $t_i \not\doteq t_j$.

This definition reflects our choice to concentrate on matching functions and assume that matching records are merged simply by taking their union. More sophisticated merge functions may be designed to find *representative values* [2] for tuple groups, but the treatment of these is outside of the scope of this paper.

2.3.2. Matching Functions

Given a record set R, a *matching function* f is a Boolean function over $R \times R$. Given records $r_1, r_2 \in R$, we write $r_1 \approx_f r_2$ to denote that $f(r_1, r_2) = true$. As is common [15], we distinguish matching functions in two classes.

Feature-level functions (also called *attribute-level*) are based on attribute-value comparisons; they are used to state that two records match each other if the values of some of their attributes are *similar*, according to some similarity notion. We intend to keep matching functions as general as possible, and therefore, we allow for notions of similarity that can be based, for example, on attribute equality or value distance. To give an example, a first matching function $f_{acronym}$ for the conference-ranking example may state that two conferences match if they have equal acronyms. A second one, f_{names}, may state that they also match if they have similar names according to a string-similarity function. Feature-level functions may involve more than one attribute to state, for example, that two events match if they have similar descriptions, the same city, and close dates.

Record-level functions are not restricted to attribute-value similarities and may employ more complex strategies to decide matches. They may employ complex statistical computations or machine learning to decide matches that go beyond simple value comparisons. For example, one might train a machine-learning model to predict the level of similarity between two records and use the output of the model to decide whether records match each other.

In the following, we shall denote by f_a and f_r feature-level and record-level functions, respectively.

2.3.3. Identifiers and Fingerprints

A crucial feature of our approach is the distinction between attributes that are used for matching purposes and those that are not. To formalize this, we denote by $X = \langle A_0, A_1, \dots, A_k \rangle$ an (ordered) list of one or more attributes. Given a record r and attributes X, the *value* $r[X]$ is the projection of r on (the alias cells of) attributes in X. Recall that when referring to values within records for the ER, we always consider the value of alias cells. Given a conference record:

$$r_1 = \langle \textit{confName:}\text{VLDB Conf.}, \textit{acronym:}\text{VLDB}, \textit{URL:}\text{http://dblp...} \rangle$$

the value $r_1[\textit{acronym}]$ is $\langle \textit{acronym:}\text{VLDB} \rangle$. Given an event record:

$$r_2 = \langle \textit{title:}\text{Music Party}, \textit{place:}\text{Bolton Str., NY}, \textit{date:}\text{May, 2014}, \textit{likes:}256 \rangle$$

and $X_{composite} = \langle title, place, date \rangle$, the value for $r[X_{composite}]$ is the tuple:

$$\langle \mathit{title}\text{:Music Party}, \mathit{place}\text{:Bolton Str., NY}, \mathit{date}\text{:May, 2014} \rangle$$

We assume that a list of attributes X_f is associated with each function f—regardless of the class f belongs to—with the property that the value of f over records r_1, r_2 only depends on $r_1[X_f], r_2[X_f]$. In addition, we require that when $r_1[X_f] = r_2[X_f]$ then $r_1 \approx r_2$. More precisely, we call the *identifying attributes* of function f (or, simply, *id* of f) the list of attributes $X_f = \langle A_0, A_1, \ldots, A_k \rangle$ such that:

- for each pair of records r_1, r_2 such that $r_1[X_f] = r_2[X_f]$, it is the case that $f(r_1, r_2) =$ *true;*
- for each combination of records r_1, r_2, r_3, r_4 such that $r_1[X_f] = r_3[X_f]$ and $r_2[X_f] = r_4[X_f]$, it is the case that $f(r_1, r_2) = f(r_3, r_4)$.

A feature-level function f_a with ID $X_{f_a} = \langle A_0, A_1, \ldots, A_k \rangle$ can be therefore rephrased as follows: records r_1, r_2 match according to f_a if, for each $i = 0, 1, \ldots k$ it is the case that $r_1[A_i]$ is "similar" to $r_2[A_i]$ (according to a notion of similarity that depends on f_a, but must be such that it preserves identity). This definition does not necessarily hold for record-level functions since they are not based on pairwise attribute comparisons, except for the equality-preserving requirement that must be true for all functions.

Given an instance I with attributes $A_0, A_i, \ldots A_k$, and a set of matching functions $f_1, f_2, \ldots f_m$, we call an *identifying attribute*, or simply *id* of I each A_i such that $A_i \in X_{f_j}$, for some matching function f_j. Any other attribute is called a *descriptive attribute*. The set of IDs of I is denoted by X_{id}.

An important tool used in our approach is *record fingerprints*. These are essentially the subsets of all ID values within a record. More formally, given a record r, and a set of IDs X_{id}, we call the *fingerprint* of r the value of $r[X_{id}]$. In the following, we will use fingerprints to store known matches and mismatches between records.

2.4. Weak Identifiers

Identifiers and their equality are at the core of our approach. Our idea is that we want to avoid the computation of a function f with ID X_f over r_1, r_2 in all cases in which $r_1[X_f] = r_2[X_f]$.

In essence, we want identifiers to be such that equal values for them guarantee that the two records match each other. This is what we call a *functional identifier*, in analogy with keys in the relational model. In symbols, we write:

$$\mathit{functional\ id}\ X : \forall r_1, r_2 : r_1[X] = r_2[X] \Rightarrow r_1 \doteq r_2$$

Examples of functional identifiers are email addresses for people and ISSN codes for journals.

Functional identifiers state that records with equal values over X are matches. Nothing can be said when their values are different. There are, however, cases in which identifiers are *total*, i.e., two records match if and only if they agree on them. Formally, X is called a *total identifier* if:

$$\mathit{total\ id}\ X : \forall r_1, r_2 : r_1[X] = r_2[X] \Leftrightarrow r_1 \doteq r_2$$

The DOI of a scientific publication and the social security number of a person are examples of total identifiers.

We know, however, that our input data may be dirty. This may cause violations of the rules above and make our identifiers become *weak ones*. Intuitively, we say that X is *weak functional identifier* (in turn, *weak total identifier*) if the formula above holds up to some probability p:

$$\mathit{weak\ functional\ id}\ X : \forall r_1, r_2 : r_1[X] = r_2[X] \Rightarrow r_1 \overset{p}{\doteq} r_2$$

$$\textit{weak total id } X : \forall r_1, r_2 : r_1[X] = r_2[X] \Leftrightarrow r_1 \overset{p}{\doteq} r_2$$

In essence, we are stating that “usually” equal values over X imply a match, although this is not necessarily true in all cases.

As we have mentioned, we do not assume that record sets are clean, i.e., they may contain errors and inconsistencies. Consequently, in our approach, identifiers are usually considered to be weak, unless it is explicitly stated otherwise as part of the input. Obviously, to make weak IDs of some interest for entity resolution, the value of p must be very high, usually above 0.9.

2.5. Checking Functions

Weak IDs may generate wrong tuple groups. In our conference-ranking example, this was due to the wrong value Int. Conf. on Management of Data associated with the COMAD conference.

Our goal in these cases is to identify the mistake using a set *check functions*, $f_{c_1}, f_{c_2}, \ldots f_{c_h}$ provided as part of the ER task specification. A *check function* f_c is a Boolean function over pairs of records. We use check functions to inspect the groups generated by the match process and accept or refuse them. More specifically, given functions $f_{c_1}, f_{c_2}, \ldots f_{c_h}$:

- we accept a group g if for each pair of records $r_1, r_2 \in g$ and each i it is the case that $f_{c_i}(r_1, r_2) = \textit{true}$;
- we refuse g as soon, for some r_1, r_2 and some i, it is the case that $f_{c_i}(r_1, r_2) = \textit{false}$.

In practice, check functions are the opposite of matching functions. While matching functions intuitively look for strong similarities among record values, check functions typically look for strong dissimilarities among record values. Each strong dissimilarity, in fact, can be considered to be a signal that the match of the two records is a false positive.

2.6. Cell Changes and Human Involvement

Recall that our purpose is to conduct the data-cleaning phase in parallel with the ER phase. Whenever we identify that a wrong group has been generated using some check function, we assume that this is due to a weak ID, i.e., to dirty values in some of the input records.

To change this, we also assume that the check function also suggests a set of *cell changes*, i.e., repair instructions for the dirty values that change them in such a way that the ER may be corrected. A *cell change* is an instruction of the form $r_{id}.A := v$, where v is a value different from the original value of cell $r_{id}.A$, possibly null.

Notice that, in many cases, finding the correct set of cell changes requires the intervention of a human expert.

In our approach, also finding matches may involve users or crowdsourcing. For example, a match function might preliminarily use automatically computed similarities to partition record comparisons in certain matches, possible matches, and non-matches and then require that humans be used to conduct a clerical review of possible matches to return the final set of matches.

The framework has been designed to be flexible in this respect. We consider match and check functions as black boxes. This leaves ample freedom to rely on human inputs in mission-critical tasks, where experience [5] tells that human decisions are the only way to achieve acceptable levels of accuracy. On the contrary, in tasks in which results of lower quality are acceptable automatic functions may be used as well.

2.7. The Match and Merge Algorithm

In this section, we present the algorithm used by Detective Gadget to implement its fast iterative entity-resolution process. The process is iterative and intuitively goes as follows (see also Figure 3):

- Phase 1—Hash-Based Matching: At each iteration, a candidate ER is generated by matching tuples via our fast hashing algorithm that reuses available knowledge about matches, typically in the form of clerical-review decisions, which we also call *user inputs*. Then, this candidate ER is subjected to further verification checks to identify false positives and false negatives.

- Phase 2—Checking for False Positives: We first look for false positives. To do this, we run check functions on matching records, looking for dissimilarities that might suggest a mismatch. Whenever these are found, we ask for additional user inputs in order to decide whether these are real mismatches or not and how to clean the input records to remove these mismatches from the next iteration. After the input data have been changed, we restart the process moving to the next iteration. It is important to note that, also in this phase, we minimize comparison by keeping track of all decisions taken so far in order to avoid the burden of re-analyzing pairs of records that are actual matches.
- Phase 3—Checking for False Negatives: As soon as phase 2 completes with no candidate false positives, we begin our search for false negatives. To identify false negatives, we run matching functions on nonmatching records, looking for similarities that might suggest a match. Whenever these are found, we ask for additional user inputs in order to decide whether these are real matches or not and move to the next iteration. Previous decisions are considered during this phase to minimize comparisons.

Both phases 2 and 3 may require interaction with expert users. The algorithm terminates as soon as both phase 2 and phase 3 return no further candidates that require inspection by human experts. It can be seen that the algorithm has refined the quality of the input datasets while performing the ER.

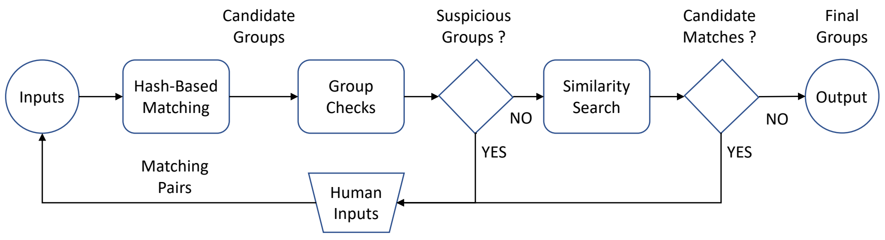

Figure 3. Detective Gadget's Workflow

Pseudocode is reported in Algorithm 1. Additional functions are in Algorithms 2–6. We can divide the pseudocode in Algorithm 1 into four main blocks, as follows.

Algorithm 1 Main Algorithm

```
Require: a set $I$ of records, matching functions $[F_a, F_r]$, checking functions $F_c$, precomputed matches/mismatches $[P_a, P_r, P'_a, P'_r]$, cell changes $C$;
Ensure: a set $G = ER(I)$ of groups of records
 1: $stop$ = false
 2: while (!$stop$) do
 3:     // Pre-processing step
 4:     applyCellChanges($I$, $C$)
 5:     aliasValues($I$, $[P_a, P_r]$)
 6:     // Hash-Based Matching step
 7:     $G \leftarrow$ fastHashMapping($I$)
 8:     // Checking for False Positives
 9:     $suspiciousGroupsFound$ = checkGroups($G$, $F_c$, $[P_a, P_r, P'_a, P'_r]$)
10:     if ($suspiciousGroupsFound$) then
11:         continue
12:     end if
13:     // Checking for False Negatives
14:     $newUserInputs$ = findSimilarities($G$, $F_c$, $[P_a, P_r, P'_a, P'_r]$)
15:     if ($newUserInputs$) then
16:         continue
17:     end if
18:     if (!$suspiciousGroupsFound$ and !$newUserInputs$) then
19:         $stop$ = true
20:     end if
21: end while
22: return $G$
```

2.7.1. Initialization

The *initialization* block declares input parameters to the algorithm. The initial input is given by:

- a collection of record sets, I, also called an *instance*;
- a set of feature-level matching functions, F_a, and a set of record-level matching functions, F_r; either of the two sets may be empty;
- a set of checking functions, F_c;
- a possibly empty set of precomputed feature-level value matches, P_a, i.e., a collection of pairs of id values $\langle v_0, v_1 \rangle$ that are considered to match each other;
- a possibly empty set of precomputed feature-level value mismatches, P'_a, i.e., a collection of pairs of ID values $\langle v'_0, v'_1 \rangle$ that are considered not to match each other;
- a possibly empty set of precomputed record-level matches, P_r, i.e., a collection of pairs of fingerprints $\langle fp_0, fp_1 \rangle$ that identify records that match each other;
- a possibly empty set of precomputed record-level mismatches, P'_r, i.e., a collection of pairs of fingerprints $\langle fp_0, fp_1 \rangle$ that identify records that do not match each other;
- a possibly empty set of cell changes C over I.

P_a, P'_a, P_r, P'_r, and C are empty at the first iteration but could not be empty in the next iterations.

After initialization, the main loop starts (lines 1–2). This will iterate operations until no new inputs are generated and no additional iterations are required.

As a first step, the algorithm applies cell changes in C to I (line 4). This will perform the value edits necessary for cleaning the input instance. Please note that whenever we edit a cell, we also edit the corresponding alias cell.

As a subsequent step, the algorithm assigns aliases to input records (Algorithm 2) to speed up the subsequent matching phase. This is based on the set of precomputed matches P_a, P_r in the input. The process is quite straightforward for attribute-level matches, and we describe it in the following. A similar yet slightly more involved technique can be used for record-level matches, i.e., fingerprints.

Algorithm 2 aliasValues

```
Require: a set $I$ of records, precomputed matches $[P_a, P_r]$
Ensure: a set $I$ with normalized alias according to $[P_a, P_r]$
1:  $valueEqvClasses$ = buildEquivalenceClasses($[P_a, P_r]$)
2:  for all $record$ in $I$ do
3:    for all $id$ in $record$ do
4:      $eqvClass$ = findClass($valueEqvClasses, r.id$)
5:      if ($eqvClass \neq null$) then
6:        $representativeValue \leftarrow$ getRepresentativeValue($eqvClass$)
7:        $alias(r.id) = representativeValue$
8:      end if
9:    end for
10: end for
```

Based on the set of precomputed ID matches P_a, we identify equivalence classes of ID values using a hashMap for each ID attribute(findClass). Then, a *representative value* is identified for each class. For each record r in I and each ID A_i of r, we find the representative value for $r[A_i]$ and assign that as the alias of $r[A_i]$. In this way, all IDs within records are aliased by the corresponding representative value. Returning to our example, assume we have established the following match $p_a \in P_a$:

$$\begin{aligned} m_1 &: \text{“ACM SIGMOD Conf. on Manag. of Data”} \\ &\approx \text{“ACM Int. Conf. on Management of Data”} \end{aligned}$$

From this match, we can infer that the two IDs belong to the same equivalence class, elect one representative for the class, and ensure that they obtain this representative value as a common alias.

2.7.2. Phase 1: Hash-Based Matching

Once aliasing has been completed, we are ready to reconstruct matches using our fast hash-based technique (line 7 of Algorithm 1). This is described in Algorithm 3.

Algorithm 3 fastHashMapping

```
Require: a set $I$ of records
Ensure: a map $M$ of records and groups
 1: $M$ = Empty double linked hash map
 2: $OM$ = Empty object map
 3: for all $record$ in $I$ do
 4:   $OM(record)$ = groupWithKey($record$, $\{record\}$)
 5: end for
 6: for all $record$ in $I$ do
 7:   for all ($id$) in $record$ do
 8:     $keyValue$ = aliasOf($id$)
 9:     $obk = OM(record)$
10:     if $M[keyValue] == null$ then
11:       $obk$.addKey($keyValue$)
12:       $M[keyValue] = obk$
13:     else
14:       mergeGroups($M$, $OM$, $keyValue$, $obk$)
15:     end if
16:   end for
17: end for
18: return $M$
```

Recall that for each matching function, f, we call X_f the set of identifying attributes of f. Recall also that, regardless of the original values in I, we have pre-processed records in order to normalize ID values by assigning the corresponding representative value as an alias. Therefore, we have the property that for any two records r_i, r_j such that there is a verified match in P_a according to ID X, the values of $r_i[X], r_j[X]$ are identical. Similarly, for any verified record match in P_r among r_k, r_l, r_k and r_l have identical fingerprints.

We leverage this property in our match phase. To do this, we use a doubly-linked hashmap that allows us to map records to the groups to which they are assigned and groups to which they need to belong to new records via their IDs. This allows us to perform the construction of candidate groups in linear time.

More precisely:

- the hash map is initialized by assigning each record r in I to the corresponding singleton group $group(r) = \{r\}$;
- then, for each matching function f in F_a or F_r, we consider the identifying attributes X_f and scan the records in I;
- for each record r, we compute $r[X_f]$, the ID of r according to X_f, and use that as a key to search the map for the corresponding group of records;
- if no group is present, we store in the map the value $group(r)$ for key $r[X_f]$;
- on the contrary, if a group $group(r[X_f])$ exists, and r does not belong to it, then we merge groups $group(r[X_f])$ and $group(r)$.

The pseudocode for the merge algorithm is in Algorithm 4. Although the actual merged group can be obtained by simply taking the union of records of the source groups, additional care needs to be put into maintaining the doubly-linked map in order to reflect the merge.

Algorithm 4 mergeGroups

```
Require: M, OM, keyValue, obk
Ensure: M, OM
1: $obk' \leftarrow$ GroupOf($M[keyValue]$) + GroupOf($obk$)
2: for all $key$ of $M[keyValue]$ + $obk$ do
3:     $obk'$.addKey($key$)
4:     $M$[key] $\leftarrow obk'$
5: end for
6: for all ($i$) of $obk'$ do
7:     OM($i$) $\leftarrow obk'$
8: end for
9: return $M, OM$
```

It is important to note that the match produced at this step correctly reflects the knowledge accumulated so far in terms of matching ID values in P_a and matching record fingerprints in P_r. In addition to being fast—even on many thousands of records, this step usually requires not more than a few seconds (as shown in Section 3)—another crucial property of the algorithm is that the match is completely reproducible based on the inputs only. Consequently, we can restart the process at each new iteration by recreating candidate matches from the start in order to incorporate any new inputs accumulated during the workflow in Figure 3.

2.7.3. Phase 2: Group Checks

Phase 1 outputs several groups that are candidates to represent the output of the ER process. These groups need to be carefully checked to identify false positives, though. In fact, in phase 1, we deliberately trade accuracy for performance: in order to have a linear-time match, we aggressively match records to each other based on IDs. Recall, however, that we assume that IDs are weak, i.e., due to dirtiness in the source data, not necessarily two records with identical IDs represent the same entity.

For this reason, in this second phase, we apply checking functions in F_c in order to verify the output of phase one and identify false positives, as follows (Algorithm 5):

- for each checking function f_c in F_c
- for each candidate group g, we check if $f_c(g)$ is true; in this case, the group is accepted and no further processing is required;
- otherwise, in case $f_c(g)$ is false, we have identified a group with potential false positives; $f_c(g)$ also returns the set of suspicious ID pairs $\langle v_i, v_j \rangle$ that triggered the alarm, call them *suspicious*(g);
- in mission-critical applications, the only way to handle these potential errors is to involve a human expert that needs to inspect *suspicious*(g) and either confirm the match or revoke it; more specifically, for each pair $\langle v_i, v_j \rangle$ in *suspicious*(g):
- if the expert confirms that the corresponding records do, in fact, match, and therefore, this is a false alarm, the value pair $\langle v_i, v_j \rangle$ is added to P_a; in this way, this specific problem will not be raised again during subsequent checks;
- if, on the contrary, those values do actually identify a false positive, the expert is supposed to generate the necessary cell change, c, in such a way that the two IDs will not match in subsequent iterations.

Please note that we always consider suspicious a group in which we have mismatching IDs, as contained in our input sets of nonmatching values, i.e., P'_a, P'_r. We assume an implicit check function devoted explicitly to this check.

If, at the end of this phase, any suspicious group is identified, additional inputs are collected from experts, inputs are updated accordingly, and the process restarts.

If, on the contrary, no suspicious group is found, we move to phase three.

In our conference example, a reasonable checking function compares the rankings of conferences in the same group. Whenever there is a significant difference, like the one between the SIGMOD and the COMAD conferences (A++ regarding B), the pair is reported as a suspicious pair. This helps to identify the incorrect match of the conference names “Int. Conf. on Management of Data”. The user needs to edit one of the conference names so that this match is no longer performed.

Algorithm 5 checkGroups

```
Require: a set $G$ of groups of records, checking functions $F_c$, precomputed
    matches/mismatches [$P_a$, $P_r$, $P'_a$, $P'_r$], cell changes $C$;
Ensure: true if suspicious groups found, false otherwise
 1: $suspiciousGroups = \emptyset$
 2: for all $group$ in $G$ do
 3:   for all $f_c$ in $F_c$ do
 4:     if $f_c(group) = false$ then
 5:       $suspiciousGroups$.add($group$) // Check failed
 6:     end if
 7:   end for
 8: end for
 9: if (!$suspiciousGroups$.isEmpty()) then
10:   askUserForInput($suspiciousGroups$, [$P_a$, $P_r$], $C$)
11:   return true
12: end if
13: return false
```

2.7.4. Phase 3: Similarity Search

After phase two, we have a collection of candidate groups that are not supposed to contain false positives but might still contain false negatives. In fact, matches so far have been based exclusively on identical ID values or on knowledge previously accumulated in the process. Still, there might be records that represent the same entity but have different IDs.

In this phase (line 14 of Algorithm 1), we run our similarity functions in order to search for additional matches (Algorithm 6). This phase may seem similar to what is carried out in other ER approaches, but we would like to emphasize two crucial differences that typically allow us to improve the performance of this step, which is by far the most expensive one in the process.

We never perform comparisons among all possible pairs of records. On the contrary, we only compare records belonging to different groups. It can be seen that this *inverted blocking* technique, in which groups are blocks that are used to prevent comparisons among their members, may in some cases significantly reduce the number of comparisons, especially in cases in which datasets contain on average a high number of duplicates and therefore groups tend to be larger.

In addition, by clearly separating the search for equality-based matches from the one for similarity-based ones, we may leverage knowledge about the collections to further speed up this step. Assume, for example, that we know that one of the recordset R has a strong identifier—for example, the DOI for research papers. In this case, we can avoid all comparisons among records of groups g_i, g_j such that g_i and g_j contain different records in R. Any possible match among records in R and their groups, in fact, has been discovered by phase 1, and no false negatives are possible.

In practice, this phase is conducted as follows:

- for each matching function f in F_a (similarly for F_r)
- we consider all pairs of distinct groups g_i, g_j in G, and all pairs of records $r_k \in g_i$, $r_l \in g_j$ for which there is no negative evidence in P'_a, P'_r
- if f states that r_k, r_l are similar, then for each attribute $A \in X[F]$ we generate a *user-input request* for the pair $\langle r_k[A], r_l[A] \rangle$; all requests are fed to human experts, which examine them

- in case an expert considers $\langle r_k[A], r_l[A] \rangle$ a match, then the pair is added to the collection of matching ID pairs, P_a
- on the contrary, $\langle r_k[A], r_l[A] \rangle$ is added to the collection of nonmatching ID pairs
- in both cases, the same comparison will be avoided in future iterations of the process.

Algorithm 6 searchSimilarities

```
Require: a set $G$ of groups of records, checking functions $F_c$, precomputed
    matches/mismatches [$P_a$, $P_r$, $P'_a$, $P'_r$], cell changes $C$;
Ensure: true if suspicious groups found, false otherwise
 1: $similarities = \emptyset$
 2: for all $g_1$ in $G$ do
 3:   for all $g_2$ in $G$ do
 4:     if $g_1 = g_2$ then
 5:       continue
 6:     end if
 7:     $similarity$ = findSimilarity($g_1$, $g_2$, [$F_a$; $F_r$])
 8:     if ($similarity \neq null$) then
 9:       $similarities$.append($similarity$)
10:     end if
11:   end for
12: end for
13: if (!$similarities$.isEmpty()) then
14:   askUserForInput($similarities$, [$P_a$, $P_r$, $P'_a$, $P'_r$])
15:   return true
16: end if
17: return false
```

If user-input requests are generated, inputs are updated, and a new iteration starts (see Figure 3). As soon as no additional user-input requests are generated by this phase, the process is concluded and the final set of groups is returned as output.

To conclude our running example, assume we are using a matching function that considers as possible matches record such that the Jaro distance [27] of confNames is above 0.75, and the edit distance between the acronyms is less than or equal to 4. The matching function would identify as possible matches record r_1 and r_4 from Figure 1 (distance of the confNames is 0.76 and the edit distance between acronyms ACM SIGMOD and SIGMOD is 4). This generates a request for user input, either on the confName or on the acronym values. In fact, similarities do not generate automatic matches in our approach. They need to be validated by users and explicitly marked as true or false positives.

3. Results and Discussion

We implemented Detective Gadget in Java. All experiments were conducted on a MacBook with an Apple M1 Max CPU@3.2Ghz and 32GB of memory.

Datasets. We used six real-world datasets representing six Dirty-ER [4] scenarios for which a golden standard is available. Some of these are classical benchmarks for entity-resolution systems. Others are based on real-life data.

1. RESTAURANTS [3] contains 864 records representing restaurants from the Fodor's and Zagat's restaurant guides that contain 112 duplicates. The attributes are name, address, city, phone, type, and class. In this context, there are no strong IDs (nor telephone and address are strong since there are records with the same telephone and address but with different names). We define two weak IDs $\langle$name, city$\rangle$, $\langle$phone$\rangle$;
2. The CDS Dataset: it contains approximately 10K CDs randomly extracted from GnuDB [4], maintained in a single file, with 299 duplicates. Attributes are pk (unique for each record), ID (unique for each record), artist, title, category, genre, year, and tracks. No strong IDs are available. We define a weak ID $\langle$artist, title, track$\rangle$;

3. The EVENTS Dataset: using an ad-hoc wrapper, we crawled events held in Italy from different sources like Facebook, Eventbrite, TicketOne, EventiESagre and TuttoCitta. Attributes are title, city, event location (address, city, facility, . . .), start date, end date, category, description, link to the web resource, poster URL, price, publisher information (name, email address, phone number, link, . . .), geographical indications (latitude and longitude). Attributes $\langle \text{title}, \text{location}, \text{date} \rangle$ represent a weak ID.
4. The CONFERENCES Dataset: such dataset contains classification that refers to the computer-science conferences area. We have considered 3 different rankings: (a) *CORE* [5] (b) *Microsoft Academic Search* [6], limited to the first 1000 conferences. (c) *LiveSHINE*. The total number of records is approximately 6K, also the gold standard contains approximately 3K groups. We are interested in four main attributes: Title, Acronym, Source, and Rank. We define two weak IDs $\langle \text{title} \rangle$ and $\langle \text{acronym} \rangle$;
5. JOURNALS Dataset: it consists of classifications of journals made by three organizations (Scopus, WOS, and ERA). Input records are approximately 70K, while the gold standard contains about 28K groups. The common information coming from the different classifications are Title (Journal name), Rank, and ISSN. We define a strong ID $\langle \text{ISSN} \rangle$ and a weak ID $\langle \text{title} \rangle$;
6. PUBLICATIONS dataset contains records from DBLP, WOS, and Scopus websites. Identified records are approximately 40K. In addition, the gold standard for publications contains about 20K groups. The common information from the different classifications are Title, Authors, DOI (optional), Venue, Year, and Citations. While the DOI is a strong ID, it is not always available. However, we define the strong ID $\langle \text{doi} \rangle$ and the weak ID $\langle \text{year}, \text{title}, \text{venue} \rangle$;

Figure 2 summarizes statistics about the size and the number of matches for each presented dataset.

Baselines and Metrics. To the best of our knowledge, there is no other entity-resolution system that is directly comparable to Detective Gadget. Therefore, our goal is to consider different approaches that may enable us to draw some comparisons, namely:

- Swoosh-based approaches, which arise from a common base;
- supervised machine-learning approaches since labels provided during the training phase can be considered similar to user inputs requested by Detective Gadget;
- non-supervised machine-learning approaches, for which it is interesting to compare the quality of results.

Based on this, we selected five baselines:

- Oyster [28], which implements the SWOOSH algorithm;
- Dedupe [29] that uses labeled examples, i.e., pairs of match or not match records, to reduce the number of inputs requested to users;
- Febrl [30] and ZeroER [31], which both use non-supervised machine learning;
- as a final baseline, we select JedAI [32], for its flexibility and variety of approaches. We tested several configurations and ultimately selected the default configuration since it is the one with the best overall results in our scenarios.

The matching functions used with Detective Gadgetare reported for each dataset in Table 1. We use *attribute-level functions*. We match two records based on their defined IDs and, more specifically, on the values of the attributes involved in the IDs. We use the following similarity functions:

- JaroWinkler [33], this metrics is useful to compare names. It privileges similarity among the prefixes of the compared strings. We use it in the direct and reverse modes to also consider similarities among the suffixes of the strings;
- SmithWaterman [34], represent the edit distance among string. This is useful to consider similarities with a small number of different characters;
- Soundex [35] uses the similarities among phonemes. It returns a high similarity when two strings have similar phonemes. This is useful for managing similarities for strings that are in languages different from English.

We match two values of the corresponding attribute if their similarity is above the reported threshold. For Equality, we check if the values are the same. For example, in the CDs dataset, we match two records if the values for artist, title, and track are similar, i.e., if, at least for each of the used similarity functions, the similarity is above the reported threshold. Considering the attributed artist, we match the values of the artist if the JaroWinkler, the SmithWaterma, or the Soundex computed similarities are greater than 0.8.

Table 1. The matching functions configuration for each dataset. For each similarity metric, we report the threshold. If two values have their similarity above the threshold are matched. - indicates that the corresponding similarity function is not defined. For Equality, we simply check if the values are the same.

		Matching Functions			
Dataset	**Attribute Name**	**Equality**	**JaroWinkler**	**SmithWaterman**	**Soundex**
CDs	artist	-	0.8	0.8	0.8
	title	-	0.8	0.8	0.8
	track	-	0.6	0.6	-
Conferences	title	-	0.76	0.8	0.66
	acronym	yes	-	-	-
Events	title	-	0.76	0.8	0.66
	city	-	0.76	0.8	0.66
	place	-	0.76	0.8	0.66
Journals	issn	yes	-	-	-
	title	-	0.8	0.9	0.6
Publications	doi	yes	-	-	-
	year	yes	-	-	-
	title	-	0.76	0.8	0.66
	venue	-	0.8	0.9	0.6
Restaurants	name	-	0.66	-	-
	city	-	0.7	0.6	0.6
	phone	-	0.6	0.6	0.6

We evaluate all systems according to various aspects: (i) quality results (precision, recall, and F-measure), (ii) execution times, (iii) user effort, and (iv) improvements of the results regarding the number of iterations. The last two points will be calculated only for the Detective Gadget system since other systems do not implement iterative ER.

To compute the precision and recall of an ER output regarding the given golden standard, we compute the two sets of pairs of matching records and compute the precision and recall of these two sets.

To measure how much Detective Gadget reduces the number of value comparisons with respect to the naive all-pairwise, quadratic approach, for each of the scenarios, we create different configurations, as follows: (i) we progressively increase the number of IDs, starting from 1 up to the maximum usable number of IDs; (ii) we start with sources that are not locally consistent, and progressively make them consistent one at a time. It should be noted that to say that a source is locally consistent is equivalent to saying that the IDs are strong in that source.

Finally, we want to emphasize that the different systems have been configured to use the same types of similarity functions and the same acceptance threshold.

Experiments

i Experiment-1 compares accuracy metrics using different systems.
ii Experiment-2 compares the execution times of the different systems.
iii Experiment-3 shows how iterations in the ER process improve the quality of the results.
iv Experiment-4 shows how previous knowledge is reused in successive iterations when the input data changes over time.

Experiment-1: Accuracy. For each dataset, we report precision (P), recall (R), and F-measure (F1) regarding the available golden standard. To make a fair comparison, all the systems were configured in order to use the same IDs. We use JedAI with the default configuration since it returns the best results.

Oyster and Febrl do not require user inputs, while Detective Gadget and Dedupe do. We limited the number of training examples for Dedupe as follows: we used the same number of user inputs as the one required by Detective Gadget when the number was lower than 1500 examples (the maximum default number for active learning in Dedupe). Since increasing the number of training examples in Dedupe above 1500 sometimes lowered quality, when the number of Detective Gadget user inputs was higher than 1500 examples, for Dedupe, we used the number of examples that gave the best quality results.

Results are reported in Table 2. The best values are in bold. It is possible to note that Febrl, Dedupe, and JedAI, in most cases, favor precision w.r.t recall, while Oyster and ZeroER favor recall regarding precision. Detective Gadget always obtains the highest F-measure. In the EVENTS scenario, Febrl reaches an accuracy close to 0, even applying different configurations. This behavior is due to the language of the dataset, which prevents the blocking algorithm employed by Febrl from working correctly. We also tried a configuration without blocking, but after 20 h, the system had not yet produced any results. Thus, the empty row in Table 2. Notice that ZeroER freezes on datasets with a high number of rows like CONFERENCES, JOURNALS, and PUBLICATIONS. Therefore, we do not report quality results for those.

Table 2. Experiment-1. Quality results. In bold the best results.

	Restaurants			**CDs**			**Events**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Oyster	0.451	0.777	0.570	0.243	0.763	0.368	0.976	0.911	0.942
Gadget	**0.966**	**1.000**	**0.982**	**1.000**	0.759	**0.863**	**1.000**	**0.999**	**1.000**
Febrl	0.611	0.491	0.545	0.935	0.629	0.752	-	-	-
Dedupe	0.924	0.866	0.894	0.941	0.690	0.796	0.999	0.978	0.988
ZeroER	0.490	0.930	0.650	0.320	**0.920**	0.470	0.030	0.020	0.024
JedAI	0.949	0.830	0.886	0.879	0.820	0.848	0.770	0.931	0.843
	Conferences			**Journals**			**Publications**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Oyster	0.906	0.522	0.570	0.142	0.575	0.228	0.995	0.646	0.783
Gadget	0.873	**0.849**	**0.861**	0.998	**0.999**	**0.999**	**0.989**	0.939	**0.963**
Febrl	0.979	0.359	0.525	**1.000**	0.701	0.824	0.921	0.867	0.893
Dedupe	**0.941**	0.687	0.794	**1.000**	0.930	0.964	0.911	0.892	0.901
ZeroER	-	-	-	-	-	-	-	-	-
JedAI	0.896	0.694	0.782	0.977	0.753	0.851	0.683	**0.977**	0.804

Experiment-2: Execution Time. Results in seconds are shown in Figure 4. For Detective Gadget, we also report the breakdown of the total time over iterations. We did not consider user think time in Dedupe and Detective Gadget.

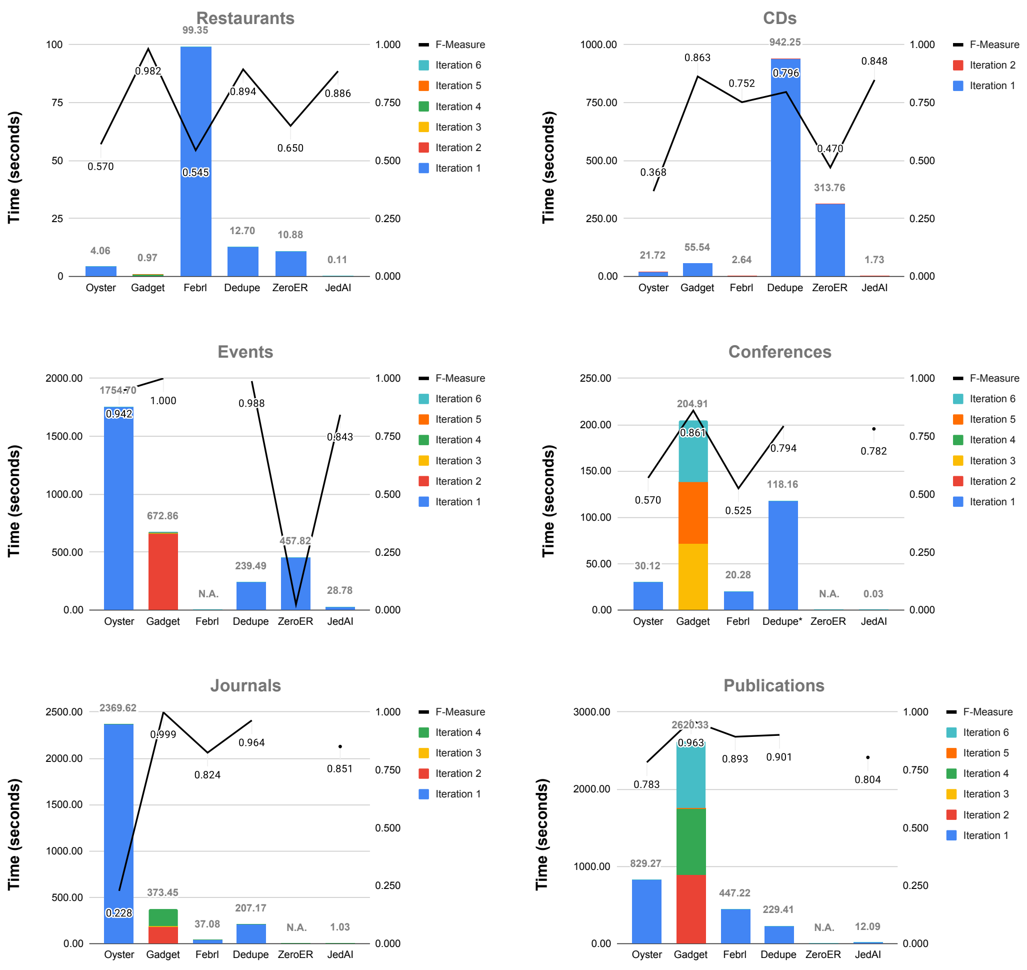

Figure 4. Execution time (in seconds) for each system. The total execution time is reported on top of each bar. N.A. indicates execution without results. The right axis reports the F-measure obtained. The * indicates the best results obtained with 1500 examples instead of the full dataset.

The first thing to note is that while Oyster, Febrl, and Dedupe use blocking, Detective Gadget does not. The second piece of evidence is that systems achieved mixed results over the various scenarios. In the Restaurant scenario, Febrl spent one order of magnitude more time than the others. Oyster is the slowest on the Events scenario. In the CDs scenario, Dedupe spent most of the time (about 800 s.) in the training phase. Conferences and Publication scenarios are the ones where Detective Gadget is the slowest system because of the high number of similar IDs that require comparisons. Notice, however, that this increased time generated better-quality results, in some cases with F-measure significantly higher than other systems. Indeed, Detective Gadget has the highest F-measures in all the scenarios.

Experiment-3: Impact of Iterations. The goal of Detective Gadget is to maximize the overall quality (F-measure) using more iterations. A drawback of the other systems is the higher number of false negatives. As we discussed in the paper, this is typically due to aggressive blocking to lower computation times.

This experiment shows how iterations in Detective Gadget help to increase the final quality by progressively removing both false positives and false negatives. Quality results are shown in Figure 5. In all six scenarios, the F-measures improved after each iteration. The major contributor is the improvement of the recall, while precision remains basically constant. Notice also that after a single iteration, Detective Gadget already has a better

F-measure regarding the baselines in four datasets (Restaurants, Conferences, Journals, and Publications).

Table 3 also reports in detail the inputs and outputs of each iteration for every scenario.

At the first iteration, Detective Gadget performs the initial matches based on identical IDs and outputs suspicious pairs of records in the same group. As can be seen from the table, the number of suspicious pairs is small at the beginning. For each of these, the user needs to inspect the data, and either mark the suspicious pair as the correct match or introduce edit rules to prevent it from further iterations. This process must be iterated for a few steps until no more suspicious pairs are generated. See, for example, Restaurants iterations 2–4 and Publications iterations 3–6.

When no more suspicious groups are found, Detective Gadget uses checking functions to find further candidate matches. The number of candidate matches depends on the thresholds used in configuring the checking functions. In real-life scenarios, we tend to use low similarity thresholds to minimize the probability of having false negatives. Thus, the number of requested user inputs may be very high. These, however, as shown in our next experiment, are typically inspected very quickly by a human expert since the vast majority—typically over 90%—represent true negatives.

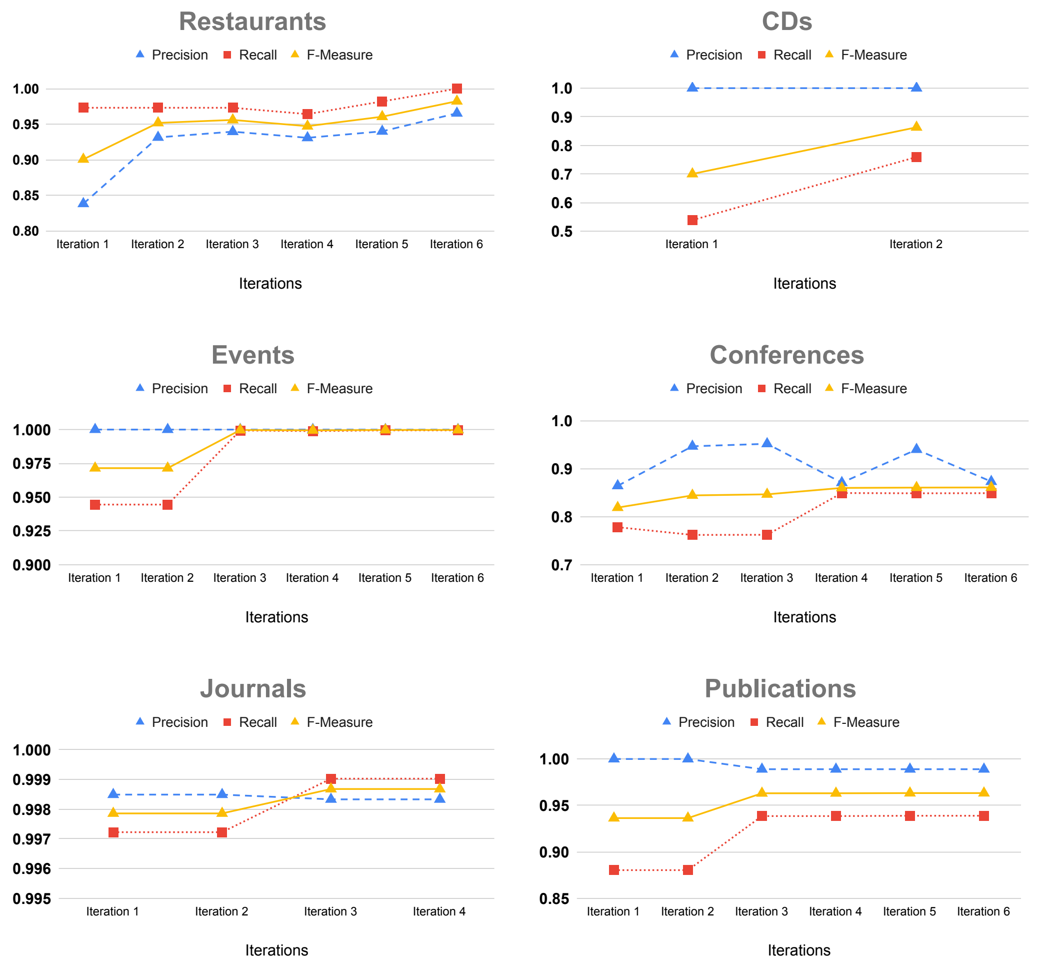

Figure 5. Quality of Detective Gadget regarding the iterations. The overall quality (F-Measure) improves at each iteration.

Table 3. Experiment-3. User Effort and Quality Details. The User Efforts is measured using the number of Verified Matches *Ver. M*, Edit Rules *Edit R.* and User Inputs *User In.* provided by the user. The outputs report the number of Suspicious *Susp* or Requested User-Input *Req* pairs to verify. Quality results are reported in terms of pair of false positive *FP*, false negative *FN*, and matching pairs from the gold standard and the ones calculated by Detective Gadget.

Scenario		**Inputs**			**Outputs**		**Quality**			
Name	**Iter.**	**Ver. M.**	**Edit R.**	**User In.**	**Susp.**	**Req.**	**FP**	**FN**	**Gold**	**Gadget**
Restaurants	1	-	-	-	13	0	21	3	112	130
Restaurants	2	6	9	0	2	0	8	3	112	117
Restaurants	3	2	0	0	1	0	7	3	112	116
Restaurants	4	0	1	0	0	2355	8	4	112	116
Restaurants	5	0	0	2355	3	0	7	2	112	117
Restaurants	6	2	1	0	0	0	4	0	112	116
CDs	1	-	-	-	0	51	0	138	299	161
CDs	2	0	0	51	0	0	0	72	299	227
Events	1	-	-	-	2	0	0	14,528	261,831	247,303
Events	2	2	0	0	0	9291	0	14,528	261,831	247,303
Events	3	0	0	9291	67	0	1	176	261,831	261,656
Events	4	63	4	0	0	45	1	296	261,831	261,536
Events	5	0	0	45	8	0	3	153	261,831	261,683
Events	6	8	0	0	0	0	5	153	261,831	261,683
Conferences	1	-	-	-	153	0	423	769	3469	3123
Conferences	2	46	116	0	6	0	148	825	3469	2729
Conferences	3	4	2	0	0	7245	133	824	3469	2778
Conferences	4	0	0	7245	8	0	436	522	3469	3383
Conferences	5	4	6	0	0	49	428	524	3469	3373
Conferences	6	0	0	59	0	0	148	825	3469	2792
Journals	1	-	-	-	194	0	130	239	86,145	83,036
Journals	2	180	0	0	0	1756	130	239	86,145	86,036
Journals	3	0	0	1756	1	0	144	84	86,145	86,205
Journals	4	1	0	0	0	0	144	84	86,145	86,025
Publications	1	-	-	-	1903	0	1	3772	31,532	27,761
Publications	2	1903	0	0	0	277	1	3772	31,532	27,761
Publications	3	0	0	277	275	0	330	1938	31,532	29,924
Publications	4	275	0	0	0	0	330	1938	31,532	29,924
Publications	5	0	0	3	7	0	330	1927	31,532	29,935
Publications	6	7	0	0	0	0	330	1927	31,532	29,935

In all scenarios, we can see that further iterations reduce errors (FP and FN) and increase accuracy so that Detective Gadget comes very close to the gold standard.

Experiment-4: Data changes over time. In this last experiment, we used a different dataset from the previous experiment. Data come from a real dataset about academic Journals, obtained by matching three databases:

- the one provided by ANVUR [7], the Italian Agency for the Evaluation of Research in Universities and Research Institutions, that rates journals in the humanities in several classes;
- the Scopus dataset of journals and indicators;
- the corresponding dataset from WOS.

All databases change every year, and the matching task can be considered a mission-critical one since it is used for research evaluation.

In this experiment, we do not report accuracy metrics since there is no clear gold standard: the dataset is too large to compute it manually. However, we can report that the resulting ER was used for 5 years in an actual system to support over 50 institutions in their research-evaluation activities, and only three errors were reported by users.

Table 4 reports the user effort in solving the ER. Each row represents an iteration. We report the date of each version of the sources and the total number of records. We also report the number of User Inputs, Edit Rules, and Verified Matches, with differences regarding the previous iteration.

Table 4. Experiment-4. User effort in solving iterative ER. Each row represents an iteration. Data sources change over time. We report the number of user inputs, edit rules and verified matches provided by the user and the time to solve them at each iteration. Most of the user effort is spent at the first iteration.

Sources		**User Inputs**			**Edit Rules**			**Verified Matches**		
Data	**Records**	**Pairs**	**Time (s)**	**Diff.**	**Rules**	**Time (s)**	**Diff.**	**Pairs**	**Time (s)**	**Diff.**
03-2017	42,756	-	-	-	8	960	+8	201	11,440	+201
03-2017	42,756	5107	6750	+5107	8	0	0	201	0	0
10-2017	43,561	5289	223	+182	8	0	0	201	0	0
02-2018	77,314	5289	0	0	6	250	−2	207	440	+6
02-2018	77,314	5353	35	+64	6	0	0	207	0	0
10-2018	36,347	5714	537	+361	6	0	0	207	0	0
03-2019	36,804	5714	0	0	6	0	0	207	0	0
05-2019	38,054	5714	0	0	6	0	0	207	0	0
09-2019	39,614	5714	0	0	6	0	0	207	0	0
11-2019	41,218	5714	0	0	6	0	0	207	0	0
04-2020	41,252	5714	0	0	10	480	+4	212	420	+5
04-2020	41,252	6284	935	+770	10	0	0	212	0	0
08-2020	44,260	6284	0	0	10	0	0	212	0	0
11-2020	41,599	6284	0	0	10	0	0	212	0	0
05-2022	41,032	6413	200	+129	10	0	0	212	0	0

We can make a few observations: (i) in some cases, multiple iterations are needed, for example, 03-2017, 02-2018 and 04-2020; (ii) most of the user effort is required in the first round of the ER (with the first version of the data of 03-2017, we solved 201 verified matches and provided 5107 user inputs). In successive iterations, even if the data changes (iii), the total time needed to solve User Inputs and Suspicious Groups is relatively low, thus confirming the effectiveness of our approach.

4. Related Work

Most proposals in the ER literature share the same basic workflow [4]: (i) A *Blocking* step, where the goal is to reduce the number of comparisons by grouping similar records into clusters to compare only records that belong to the same group; (ii) A *Matching* step, where records in the same cluster are compared using matching functions; (iii) A *Clustering* step, typically optional, which searches for additional, less probable matches.

The main differences between this typical workflow and the Detective Gadget workflow are in the workflow itself and the pre-processing step. First, the Detective Gadget workflow is iterative, as depicted in Figure 3. In addition, while in all previous works, the data-cleaning process [22] is carried out before the ER starts and, therefore, it is considered to be a pre-processing step, using either automatic approaches [24] or human-in-the-loop [25,36], in Detective Gadget data-cleaning and ER activities are intertwined.

JedAI [32,37] is a tool that implements three different workflows:

- Blocking-based: that uses blocking to merge similar entities and reduce the search space using different blocking functions. Then, records in the same block are compared using string-similarity functions to partition and split the data into equivalence clusters, i.e., records that refer to the same entity. The workflow also contains optional steps that can be used to improve the performance. For example, schema clustering improves the precision by grouping similar attributes exploiting the similarity between the names or the values. Instead, block cleaning reduces the number of comparisons discarding blocks with a high number of records. Intuitively, the bigger the group, the higher the probability that such a group contains different entities.
- Join-based: it uses matching rules to find similar records. A matching rule contains similarity metrics, the attributes to which such metrics are applied, and a threshold to define the similarity.
- Progressive: is an extension of the blocking-based pipeline that introduces a limit in the comparisons. To achieve good results and reduce the number of comparisons a prioritization step is introduced to the processing order such that the matching records are detected sooner. The prioritization step is introduced before the blocking comparisons.

Compared with Detective Gadget, the most similar workflow is the blocking-based that corresponds to the default configuration used in our experiments.

Depending on the settings of the ER problem there are a plethora of ER systems already developed. These systems can be classified as Learning-Based and Not-Learning-Based [4], i.e., systems that accept external information like human intervention and systems that use data only. Detective Gadget is meant to work in a Learning-based mode and typically relies on human interactions, but it can also be used in a Not-Learning mode by appropriately choosing the matching and checking functions and by properly implementing the cell-change generation module.

In the remainder of this section, we consider a few proposals in the literature and discuss their comparison to Detective Gadget.

SWOOSH [15] introduced an abstract framework for the study of ER algorithms and two algorithms, called R-SWOOSH and F-SWOOSH, for record-level and attribute-level comparisons, respectively. SWOOSH is abstract in the sense that it makes very limited assumptions about the actual match and merge functions, which are considered black boxes. Assuming the functions have a number of natural properties, it was proven that R-SWOOSH is optimal in the number of record comparisons it performs. Experimental evaluations have shown [5,15] that, while record-level functions may, in some cases, give outputs of higher quality, they are considerably slower than attribute-level approaches. In fact, experiments show that F-SWOOSH is significantly faster than R-SWOOSH. Oyster [38] system uses the R-SWOOSH algorithm. It also uses blocking, offering classical techniques like Scan, Soundex, and DMSoundex, but it also offers the possibility to specify a composite index for blocking.

Is it possible to reduce the number of comparisons by reducing the set of candidate records, speeding up a factor up to 2.6×–5× over previous algorithms [39]? This algorithm, unlike our approach, is based only on reducing the number of comparisons, adopting a technique based on the ordering of the tokens in the record and combining it with techniques based on prefixes. In our case, we rely on a hierarchy of ID and on an iterative process that allows us to clean the data as they are grouped with references to the same entity.

Other systems [20,40] try to manage the problem of the comparisons of a very large number of records by distributing the problem. However, these approaches rely heavily on the use of multiple processors to solve the problem but do not deviate much from the classic approach of SWOOSH.

From the point of view of the human user involvement in the process of ER, CrowdER [11] is an approach in this direction. The idea is to give the possibility to present to the user requests for the records that are over a certain similarity threshold. The previous work has proposed batching verification tasks for presentation to human workers, but even with batching, a human-only approach is infeasible for data sets of even moderate size due to the large numbers of matches to be tested. In this approach, the intuition is that crowdsourcing platforms are increasingly used but, although useful, are expensive. So, it is presented as a system capable of producing user requests as efficiently as possible in order to spend the least amount of resources (money), minimizing the use of different workers. Magellan [41] allows users to explore and clean the data before the ER process. The user is guided through a how-to guide to develop an ER pipeline. Magellan differs from Detective Gadgetdue to the fact that the cleaning and ER process is not iterative. Corleone [42] uses the crowd to solve the complete ER. Dedupe [29] is a system that uses machine learning. It also makes use of clustering using a hierarchical method. To reduce the number of labeled examples it uses active learning. A training example is composed of a pair of records identified with the declared variables, and the possible labels are yes, no, or not sure.

Febrl [30] can be used in learning and not-learning mode. In the learning mode, a classifier extracts weights from a highly similar record and uses those weights as features for training an SVM classifier. In the not-learning mode, Febrl makes use of clustering algorithms to find similar records.

ZeroER [31] uses generative models to calculate if two records match or not. The parameters of the model are estimated using the EM algorithm [43] from some generated features without training data, i.e., without human intervention. Such features are calculated using various functions (e.g., similarity, exact match, . . .). The main idea is that two records that are similar share similar feature vectors.

SystemER [44] generates Boolean features from the attributes. Then, it uses active learning to propose that the user manually annotate the likely false positives and the likely false negatives. We do not compare SystemER with Detective Gadget since the source code is not available.

Recently, some proposals have used Deep Learning (DL). Ditto [45] is an ER solution that leverages pre-trained language models (PLMs) based on transformer architecture. It models the ER task as a classification task on pairs on record. It allows the inclusion of domain knowledge at the blocking level, where the user can specify attributes that are equivalent to our "id attributes" or replace similar values like our "alias". It also uses summarization on long strings to compare similarity and augmenting techniques to learn from difficult examples. Essentially, Ditto uses DL at the matching level. DL could also be used at blocking level [46] with different solutions. While previous DL solutions are domain-specific, i.e., they are dataset-specific and to be used on a new dataset, they should be finetuned with at least hundreds of examples, Unicorn [47] is a novel system that removes domain limitations and works in a setting with zero-shot prediction using a prompt provided to the PLMs. The prompt provided to the PLM is a question, "Does x match with y?", where x and y are linearizations or the two records.

Finally, there are new approaches based on the so-called Progressive ER, where the goal is to produce a partial solution of the ER [48,49]. These approaches have in common the need to reiterate the ER process whenever the sources are updated, but while Detective Gadget generates a complete solution, their goal is to give the best solution with a limited budget (time or computation).

5. Conclusions

To the best of our knowledge, Detective Gadget represents the first proposal of a structured workflow to handle mission-critical ER tasks where human experts are involved.

The main idea behind the system is to structure the process iteratively by clearly separating the search for matches based on identical values from verification steps that spot potential false positives and false negatives while systematically involving humans.

We believe that this paper makes two main contributions to the field of semi-automatic ER:

- on the one side, it shows that our workflow can actually achieve the level of quality typically requested in mission-critical ER tasks;
- on the other side, it proves that our separation of concerns often guarantees the best trade-off between quality and computation times.

We believe that these are important advancements toward the goal of adopting ER algorithms in practical, mission-critical tasks. In these tasks, automatic systems in the literature—which represent a good solution to scenarios of a different kind, where quality is less critical—have so far performed poorly. On the one side, they are typically unable to guarantee the desired level of accuracy. On the other side, they provide clear guidance on how to involve human experts and perform clerical reviews of results.

Detective Gadgethas been used to handle real-life, mission-critical ER scenarios, all in the context of the development of an information system for research evaluation in Italy. We mention a few of these and report some stats:

- a large ER task about journal classifications has been handled; the task involves 22 different journal classifications, for a total of over 48,000 records; records are matched to roughly 20,000 journals; the task was developed incrementally over several months by progressively adding new classifications; we do not have precise worktime logs, but we estimate that the total worktime to complete the task has been approximately 10 work hours;
- an ER task about computer-science conference ratings has been performed in order to develop the GGS International Computer-Science Conference Rating (https://scie.lcc.uma.es/ (accessed on 18 November 2024)); the task involves three major computer-science conference ratings for a total of over 4000 records; records are matched to roughly 1600 conferences; the task was developed incrementally over several months by progressively adding new classifications;
- an ER task about "author IDs" in the main bibliometric databases, Scopus and WOS, is ongoing for approximately 35,000 Italian researchers; the goal is to identify their unique identifiers within the Scopus and WOS database in order to properly compute self-citations, one of the criteria in the Italian national evaluation procedures. Interestingly, Scopus and WOS also use (semi-)automatic ER techniques to match author IDs to researchers, and experience tells that the quality of results varies greatly from one scientific field to the other, and inconsistencies are very common.

In all these approaches, Detective Gadgethas guaranteed an excellent trade-off between the quality of results—a crucial requirement in research evaluation—and effort.

As future work, we envision three main research directions. First, we believe that additional optimizations in the similarity-search phase can be introduced by developing forms of reasoning that automatically characterize IDs in recordsets, even when a priori knowledge about these is available. On the other side, we believe that a more sophisticated user interface supporting the ER task configuration through the definition of the data sources and the matching and checking function but also to allow collaborative work and control of the phases of the process by administrators and human experts might further improve the effectiveness of the system. As a final research direction, we believe that we can leverage Large Language Models [50] to reduce the number of both user interventions and the number of human experts using the In-Context Learning approach to avoid hallucination problems.

Author Contributions: Conceptualization, M.B. and G.M.; methodology, G.M.; software, M.B., G.M., D.S. and E.V.; validation, M.B., D.S. and E.V.; formal analysis, D.S. and E.V.; investigation, M.B. and D.S.; writing—original draft preparation, M.B. and G.M.; writing—review and editing, G.M., D.S. and E.V.; visualization, E.V.; supervision, G.M.; project administration, G.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is available after request to the authors.

Conflicts of Interest: Author Marcello Buoncristiano was employed by the company Svelto!. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Notes

[1] http://www.core.edu.au/ (accessed on 18 November 2024).

[2] http://dblp.uni-trier.de/ (accessed on 18 November 2024).

[3] https://www.cs.utexas.edu/users/ml/riddle/ (accessed on 18 November 2024)

[4] https://gnudb.org/ (accessed on 18 November 2024).

[5] http://www.core.edu.au/ (accessed on 18 November 2024).

[6] https://www.microsoft.com/en-us/research/project/academic/ (accessed on 18 November 2024).

[7] https://www.anvur.it/en/homepage/ (accessed on 18 November 2024).

References

1. Chang, C.H.; Kayed, M.; Girgis, M.R.; Shaalan, K.F. A Survey of Web Information Extraction Systems. *IEEE Trans. Data Know. Eng.* **2006**, *18*, 1411–1428. [CrossRef]
2. Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S. Duplicate Record Detection: A Survey. *IEEE Trans. Data Know. Eng.* **2007**, *19*, 1–16. [CrossRef]
3. Christen, P. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*; Springer: Berlin/Heidelberg, Germany, 2012.
4. Papadakis, G.; Ioannou, E.; Thanos, E.; Palpanas, T. The Four Generations of Entity Resolution. *Synth. Lect. Data Manag.* **2021**, *16*, 1–170.
5. Köpcke, H.; Thor, A.; Rahm, E. Comparative Evaluation of Entity Resolution Approaches with FEVER. *PVLDB* **2009**, *2*, 1574–1577. [CrossRef]
6. Lapadula, P.; Mecca, G.; Santoro, D.; Solimando, L.; Veltri, E. Humanity Is Overrated. or Not. Automatic Diagnostic Suggestions by Greg, ML (Extended Abstract). In *Communications in Computer and Information Science, Proceedings of the New Trends in Databases and Information Systems—ADBIS 2018 Short Papers and Workshops, AI*QA, BIGPMED, CSACDB, M2U, BigDataMAPS, ISTREND, DC, Budapest, Hungary, 2–5 September 2018*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 909, pp. 305–313. [CrossRef]
7. Lapadula, P.; Mecca, G.; Santoro, D.; Solimando, L.; Veltri, E. Greg, ML–Machine Learning for Healthcare at a Scale. *Health Technol.* **2020**, *10*, 1485–1495. [CrossRef]
8. Sagiroglu, S.; Sinanc, D. Big data: A review. In Proceedings of the 2013 International Conference on Collaboration Technologies and Systems (CTS), San Diego, CA, USA, 20–24 May 2013; pp. 42–47.
9. Glavic, B.; Mecca, G.; Miller, R.J.; Papotti, P.; Santoro, D.; Veltri, E. Similarity Measures For Incomplete Database Instances. In Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, 25–28 March 2024; pp. 461–473. [CrossRef]
10. Fan, W.; Geerts, F. *Foundations of Data Quality Management*; Morgan & Claypool: San Rafael, CA, USA, 2012.
11. Wang, J.; Kraska, T.; Franklin, M.J.; Feng, J. CrowdER: Crowdsourcing Entity Resolution. *Proc. VLDB Endow.* **2012**, *5*, 1483–1494. [CrossRef]
12. Lee, J.; Cho, H.; Park, J.; Cha, Y.; Hwang, S.; Nie, Z.; Wen, J. Hybrid Entity Clustering Using Crowds and Data. *VLDB J.* **2013**, *22*, 711–726. [CrossRef]
13. Veltri, E.; Badaro, G.; Saeed, M.; Papotti, P. Data Ambiguity Profiling for the Generation of Training Examples. In Proceedings of the 39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, 3–7 April 2023; pp. 450–463. [CrossRef]
14. Veltri, E.; Santoro, D.; Badaro, G.; Saeed, M.; Papotti, P. Pythia: Unsupervised Generation of Ambiguous Textual Claims from Relational Data. In *Proceedings of the SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, 12–17 June 2022*; Ives, Z.G., Bonifati, A., Abbadi, A.E., Eds.; ACM: New York, NY, USA, 2022; pp. 2409–2412. [CrossRef]
15. Benjelloun, O.; Garcia-Molina, H.; Menestrina, D.; Su, Q.; Whang, S.E.; Widom, J. Swoosh: A Generic Approach to Entity Resolution. *VLDB J.* **2009**, *18*, 255–276. [CrossRef]
16. Giuzio, A.; Mecca, G.; Quintarelli, E.; Roveri, M.; Santoro, D.; Tanca, L. INDIANA: An interactive system for assisting database exploration. *Inf. Syst.* **2019**, *83*, 40–56. [CrossRef]
17. Binette, O.; Steorts, R.C. (Almost) all of entity resolution. *Sci. Adv.* **2022**, *8*, eabi8021. [CrossRef]
18. Verroios, V.; Garcia-Molina, H. Entity resolution with crowd errors. In Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, Seoul, Republic of Korea, 13–17 April 2015; pp. 219–230.

19. Konstantinos, N.; Ioannou, E.; Papadakis, G. The Five Generations of Entity Resolution on Web Data. In Proceedings of the International Conference on Web Engineering, Tampere, Finland, 17–20 June 2024.
20. Kolb, L.; Thor, A.; Rahm, E. Dedoop: Efficient Deduplication with Hadoop. *Proc. VLDB* **2012**, *5*, 1878–1881. [CrossRef]
21. Chu, X.; Ilyas, I.F.; Krishnan, S.; Wang, J. Data cleaning: Overview and emerging challenges. In Proceedings of the 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 2201–2206.
22. Ilyas, I.F.; Chu, X. *Data Cleaning*; Morgan & Claypool: San Rafael, CA, USA, 2019.
23. Chu, X.; Ilyas, I.F.; Papotti, P. Holistic data cleaning: Putting violations into context. In Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, QLD, Australia, 8–12 April 2013; pp. 458–469.
24. Geerts, F.; Mecca, G.; Papotti, P.; Santoro, D. Cleaning data with LLUNATIC. *VLDB J.* **2020**, *29*, 867–892. [CrossRef]
25. He, J.; Veltri, E.; Santoro, D.; Li, G.; Mecca, G.; Papotti, P.; Tang, N. Interactive and deterministic data cleaning. In Proceedings of the 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 893–907.
26. Whang, S.E.; Benjelloun, O.; Garcia-Molina, H. Generic Entity Resolution with Negative Rules. *VLDB J.* **2009**, *18*, 1261–1277. [CrossRef]
27. Jaro, M.A. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *J. Am. Stat. Assoc.* **1989**, *84*, 414–420. [CrossRef]
28. Talburt, J.R.; Zhou, Y. A practical guide to entity resolution with OYSTER. In *Handbook of Data Quality*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 235–270.
29. Forest, G.; Derek, E. Dedupe. 2019. Available online: https://github.com/dedupeio/dedupe (accessed on 18 November 2024).
30. Christen, P. Febrl—An open source data cleaning, deduplication and record linkage system with a graphical user interface. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV, USA, 24–27 August 2008; pp. 1065–1068.
31. Wu, R.; Chaba, S.; Sawlani, S.; Chu, X.; Thirumuruganathan, S. Zeroer: Entity resolution using zero labeled examples. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Portland, OR, USA, 14–19 June 2020; pp. 1149–1164.
32. Papadakis, G.; Tsekouras, L.; Thanos, E.; Giannakopoulos, G.; Palpanas, T.; Koubarakis, M. The return of jedai: End-to-end entity resolution for structured and semi-structured data. *Proc. VLDB Endow.* **2018**, *11*, 1950–1953. [CrossRef]
33. Wang, Y.; Qin, J.; Wang, W. Efficient approximate entity matching using jaro-winkler distance. In *Proceedings of the International Conference on Web Information Systems Engineering*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 231–239.
34. Pearson, W.R. Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics* **1991**, *11*, 635–650. [CrossRef]
35. Holmes, D.; McCabe, M.C. Improving precision and recall for soundex retrieval. In Proceedings of the Proceedings. International Conference on Information Technology: Coding and Computing, Las Vegas, NV, USA, 8–10 April 2002; pp. 22–26.
36. Mecca, G.; Papotti, P.; Santoro, D.; Veltri, E. BUNNI: Learning Repair Actions in Rule-driven Data Cleaning. *ACM J. Data Inf. Qual.* **2024**, *16*, 12:1–12:31. [CrossRef]
37. Papadakis, G.; Mandilaras, G.; Gagliardelli, L.; Simonini, G.; Thanos, E.; Giannakopoulos, G.; Bergamaschi, S.; Palpanas, T.; Koubarakis, M. Three-dimensional entity resolution with JedAI. *Inf. Syst.* **2020**, *93*, 101565. [CrossRef]
38. Zhou, Y.; Talburt, J.; Su, Y.; Yin, L. OYSTER: A tool for entity resolution in health information exchange. In Proceedings of the 5th International Conference on Cooperation and Promotion of Information Resources in Science and Technology, Beijing, China, 27–29 November 2010; pp. 358–364.
39. Xiao, C.; Wang, W.; Lin, X.; Yu, J.X.; Wang, G. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst. TODS* **2011**, *36*, 15. [CrossRef]
40. Benjelloun, O.; Garcia-Molina, H.; Gong, H.; Kawai, H.; Larson, T.E.; Menestrina, D.; Thavisomboon, S. D-swoosh: A family of algorithms for generic, distributed entity resolution. In Proceedings of the Distributed Computing Systems, ICDCS'07, Toronto, ON, Canada, 25–27 June 2007; p. 37.
41. Konda, P.; Das, S.; Doan, A.; Ardalan, A.; Ballard, J.R.; Li, H.; Panahi, F.; Zhang, H.; Naughton, J.; Prasad, S.; et al. Magellan: Toward building entity matching management systems over data science stacks. *Proc. VLDB Endow.* **2016**, *9*, 1581–1584. [CrossRef]
42. Gokhale, C.; Das, S.; Doan, A.; Naughton, J.F.; Rampalli, N.; Shavlik, J.; Zhu, X. Corleone: Hands-off crowdsourcing for entity matching. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, UT, USA, 22–27 June 2014; pp. 601–612.
43. Dempster, A.P.; Laird, N.M.; Rubin, D.B. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc. Ser. B Methodol.* **1977**, *39*, 1–22. [CrossRef]
44. Qian, K.; Popa, L.; Sen, P. Systemer: A human-in-the-loop system for explainable entity resolution. *Proc. VLDB Endow.* **2019**, *12*, 1794–1797. [CrossRef]
45. Li, Y.; Li, J.; Suhara, Y.; Doan, A.; Tan, W. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* **2020**, *14*, 50–60. [CrossRef]
46. Thirumuruganathan, S.; Li, H.; Tang, N.; Ouzzani, M.; Govind, Y.; Paulsen, D.; Fung, G.; Doan, A. Deep Learning for Blocking in Entity Matching: A Design Space Exploration. *Proc. VLDB Endow.* **2021**, *14*, 2459–2472. [CrossRef]
47. Tu, J.; Fan, J.; Tang, N.; Wang, P.; Li, G.; Du, X.; Jia, X.; Gao, S. Unicorn: A Unified Multi-tasking Model for Supporting Matching Tasks in Data Integration. *Proc. ACM Manag. Data* **2023**, *1*, 84:1–84:26. [CrossRef]

48. Papenbrock, T.; Heise, A.; Naumann, F. Progressive duplicate detection. *IEEE Trans. Knowl. Data Eng.* **2014**, *27*, 1316–1329. [CrossRef]
49. Simonini, G.; Papadakis, G.; Palpanas, T.; Bergamaschi, S. Schema-agnostic progressive entity resolution. *IEEE Trans. Knowl. Data Eng.* **2018**, *31*, 1208–1221. [CrossRef]
50. Chang, Y.; Wang, X.; Wang, J.; Wu, Y.; Yang, L.; Zhu, K.; Chen, H.; Yi, X.; Wang, C.; Wang, Y.; et al. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.* **2024**, *15*, 1–45. [CrossRef]