Query Determined Methodology to Article Resolution

Hotham Altwaijry   Dmitri V. Kalashnikov   Sharad Mehrotra

Department of Computer Science
University of California, Irvine

ABSTRACT

This paper explores "on-the-fly" data cleaning in the context of a user query. A novel *Query-Driven Approach (QDA)* is developed that performs a minimal number of cleaning steps that are only necessary to answer a given selection query correctly. The comprehensive empirical evaluation of the proposed approach demonstrates its significant advantage in terms of efficiency over traditional techniques for query-driven applications.

1. INTRODUCTION

The significance of data quality research is motivated by the observation that the effectiveness of data-driven technologies such as decision support tools, data exploration, analysis, and scientific discovery tools is closely tied to the *quality of data* to which such techniques are applied. It is well recognized that the outcome of the analysis is only as good as the data on which the analysis is performed. That is why today organizations spend a substantial percentage of their budgets on cleaning tasks such as removing duplicates, correcting errors, and filling missing values, to improve data quality prior to pushing data through the analysis pipeline.

Given the critical importance of the problem, many efforts, in both industry and academia, have explored systematic approaches to addressing the cleaning challenges [4, 11]. In this paper we focus primarily on the *entity resolution* challenge that arises when objects in the real world are referred to using references or descriptions that are not always unique identifiers of the objects, leading to ambiguity.

Efficiency, along with quality, has always been key traditional challenges of entity resolution. Entity resolution can be computationally expensive, for instance, it may require $O(n^2)$ calls to *resolve* functions (explained in Sec. 4) per each cleaning block of size n. A resolve function, in general, may require complex graph analysis, ontology matching, database lookup, consulting external data sources, and/or seeking human input through crowdsourcing [12, 21] to compute similarity between two input records and thus can be very expensive.

Traditionally, data cleaning is performed as a preprocessing step when creating a data warehouse prior to making it available to analysis – an approach that works well under standard settings. Cleaning the entire data warehouse, however, can require a considerable amount of time and significant computing resources. Hence, such an approach is often suboptimal for many modern query-driven applications that need to analyze only small portions of the entire dataset and produce answers "on-the-fly" and in real-time [7, 18].

A query-driven approach is motivated by several key perspectives. The primary motivation of this paper is queries on *online data*. For example, queries on Google Scholar, Google Fusion Tables, Yelp, Zagat, Amazon, eBay, extensions to Web People Search, etc. The idea is that of meta-search technology: a middleware system is able to answer `SQL`-like queries, issued by the user, on top of one or multiple online sources, while de-duplicating results in the process. Similar types of entity resolution systems have already been considered, e.g. in [26], where the system will know "what to clean" only at query time, thus it cannot clean data beforehand. The difference between our approach and the approach in [26] is that our solution considers *semantics* of the given query and thus can process certain types of queries much more efficiently. A query-driven approach is also useful when a *small* organization is in possession of a very large dataset, but typically needs to analyze only small portions of it to answer some analytical queries quickly. In this case, it would be counterproductive for this organization to spend their limited computational resources on cleaning all the data, especially given that most of it is going to be unnecessary. Another scenario where a query-driven approach could be useful is in the context of streaming (web) data, e.g., data from social media such as tweets from Twitter. A storage of such *fast* data and then cleaning it all as in a warehouse setting might not be feasible or practical – as limited computing resources might be dedicated to more important tasks. Handling of streaming data, however, has several nuances which are not considered in this paper.

To address these new cleaning challenges, in this paper we propose a *Query-Driven Approach (QDA)* to data cleaning. Currently, the main traditional approach for improving the efficiency of ER is that of *blocking.* QDA is an entirely new complementary paradigm for improving the efficiency: it is different from blocking and, as we will show, is typically much more effective in conjunction with blocking. However, unlike blocking, QDA is not generic as it is meant primarily for query-driven tasks. QDA exploits the specificity and semantics of the given `SQL` selection query to significantly reduce the cleaning overhead by resolving only those records that may influence the query's answer. It computes answers that are equivalent to those obtained by first using a regular cleaning algorithm, e.g., [5,11], and then querying on top of the cleaned data. However, we will see that in many cases QDA can compute these answers much more efficiently.

A key concept driving the QDA approach is that of *vestigiality.* A cleaning step (i.e., call to the resolve function for a pair of records) is called vestigial (redundant) if QDA can guarantee that it can still compute a correct final answer without knowing the outcome of this resolve. We formalize the concept of vestigiality in the context of a large class of SQL selection queries and develop techniques to identify vestigial cleaning steps. Technical challenges arise since vestigiality, as we will show, depends on several factors, including the specifics of the cleaning function (e.g., the merge function used if two objects are indeed duplicate entities), the predicate associated with the query, and the query answer semantics of what the user expects as the result of the query. We show that determining vestigiality is NP-hard and we propose an effective approximate solution to test for vestigiality that performs very well in practice.

The **main contributions** of this paper are:

- Introduction of the query-driven ER problem that systematically exploits semantics of query predicates to reduce overhead of data cleaning. We believe ours is the first paper to explore such a concept in a systematic manner in the context of SQL selection queries (Sec. 3 and 4).
- Introduction of the concept of *vestigiality* of certain computations in the context of a solution for SQL selection queries (Sec. 5).
- Development of query-driven techniques that leverage the concept of vestigiality to reduce computation (Sec. 6).
- Extensive empirical evaluation of QDA. (Sec. 7).

The rest of this paper is organized as follows. Section 2 covers the related work. A motivating example is presented in Section 3. The problem definition is provided in Section 4. Section 5 explains the concept of vestigiality. Our solution is described in Section 6 and tested in Section 7. Finally, we conclude the paper in Section 8.

2. RELATED WORK

Entity resolution is a well-known problem and it has received significant attention in the literature over the past few decades. A thorough overview of the existing work in this area can be found in surveys [13]. We classify the ER techniques into two categories as follow:

Generic ER. A typical ER cycle consists of several phases of data transformations that include: *normalization*, *blocking*, *similarity computation*, *clustering*, and *merging* [17], which can be intermixed.

In the *normalization* phase, the ER framework standardizes the data formats. The next phase is *blocking* which is a main traditional mechanism used for improving ER efficiency [16]. Often blocking partitions records into buckets [19] or canopies [22]. After that, in the *similarity computation* phase, the ER framework uses a *resolve/similarity function* to compute the similarity between the different real-world entities. Traditional methods analyze the similarity of entities to determine if they co-refer [15, 16, 23]. Recently new approaches exploit new information sources such as analyzing context [4, 9, 29], exploiting relationships between entities [20], domain/integrity constraints [14], behaviors of entities [28], and external knowledge bases such as ontologies and web search engines [12, 21, 24]. The next ER phase is *clustering* where matching records are grouped together into clusters [5,11]. Finally, the *merging phase* combines elements of each individual cluster into a single record.

On-the-fly ER. On-the-fly matching techniques have been proposed in [6, 18, 25]. The approach in [6] answers queries collectively using a two-phase "expand and resolve" algorithm. It retrieves the related records for a query using two expansion operators, and then answers the query by only considering the extracted records. An example of a query is to retrieve *all* papers written by author 'J. Smith'. Unlike our work, that paper does not consider optimizing for other types of selection queries such as range queries or queries where the type of the condition attribute is not a string.

Even though the ER technique in [18] is also "on-the-fly", it solves a different problem since it resolves queries under data uncertainty by connecting ideas of record linkage and probabilistic databases. The term *query* refers to a combination of (attribute-name/value) pairs and each entity returned as an answer is accompanied by a probability that this entity will be selected amongst all possible worlds.

In [25], the authors handle entity uncertainty at query-time for OLAP applications. Unlike ours, this work assumes the existence of a record-to-cluster mapping table and its goal is to answer group-by OLAP queries by returning results in the form of strict ranges.

Note that the approaches in [6,18,25] cannot answer generic selection queries like: select *only* well-cited (e.g., with citation count above 45) papers written by 'J. Smith' – which is the primary focus of our paper. That is, none of the existing solutions consider optimizing generic SQL selection queries studied in our paper.

3. MOTIVATING EXAMPLE

Before formalizing the notion of vestigiality and developing QDA, in this section we describe the main idea behind our query-driven solution using an illustrative example.

Setup. Consider a user searching for bibliographic information of a researcher named "Alon Halevy" in Google Scholar, the results of which are shown in Table 1[1]. Let us further assume that all the publications are written by the same author, but some of the papers returned could be duplicates. Such duplicates possibly arise as a result of the method Google uses to get its information: Google crawls research publications and then extracts citations. Since the same paper might be referred to differently in different publications, paper clusters could be split in parts, leading to duplication. In Table 1, papers $\{p_1, p_7\}$, $\{p_2, p_3, p_4\}$, and $\{p_5, p_6\}$ are duplicates and refer to the same real-world entities. Hence, they should be clustered into three clusters C_1, C_2 and C_3 by an ER algorithm.

Suppose that the user is actually *not* interested in *all* papers of "Alon Halevy", but only in well-cited ones, e.g., those with a citation count above or equal to, say, 45. The following query represents the user's area of interest:

Query 1. SELECT $*$ FROM R WHERE $cited \geq 45$

When Query 1 is issued on Table 1 prior to cleaning it, the results are p_1 and p_7, corresponding to cluster C_1. This

[1]This running example is synthetically generated and is only used for illustration purposes.

p_id	p_title	cited	venue	authors	year
p_1	Towards efficient entity resolution	65	Very Large Data Bases	Alon Halevy	2000
p_7	Towards efficient ER	45	VLDB	Alon Halevy	2000
p_2	Entity Resolution on dynamic data	25	ACM SIGMOD	Alon Halevy, Jane Doe	2005
p_3	ER on dynamic data	20	Proc of ACM SIGMOD Conf	A. Y. Halevy, J. Doe	2005
p_4	Entity-Resolution for dynamic data	15	SIGMOD Conf	A. Halevy, Jane D.	2005
p_5	Entity-Resolution for census data	10	ICDE Conf	Alon Halevy	2002
p_6	ER on census data	5	Proc of ICDE Conf	Alon Y. Halevy	2002

Table 1: Relation R. Some papers are duplicates. $C_1 = \{p_1, p_7\}$, $C_2 = \{p_2, p_3, p_4\}$ and $C_3 = \{p_5, p_6\}$ are 3 clusters.

cluster	p_id	p_title	cited	venue	authors	year
C_1	$p_1 \oplus p_7$	Towards efficient entity resolution	110	Very Large Data Bases	Alon Halevy	2000
C_2	$p_2 \oplus p_3 \oplus p_4$	Entity-Resolution on dynamic data	60	Proc of ACM SIGMOD Conf	Alon Halevy, Jane Doe	2005
C_3	$p_5 \oplus p_6$	Entity-Resolution for census data	15	Proc of ICDE Conf	Alon Halevy	2002

Table 2: Clustering $\mathcal{C}$: relation R after being clustered using TC.

is incorrect since the second paper cluster C_2 has a citation count equal to $60 \geq 45$ and should also be returned.

Standard Solution. The standard way to answer Query 1 is to first deduplicate relation R to create merged profiles of each paper and then compute the query over this clustering. Suppose that we use a variant of the transitive closure (TC) algorithm for this purpose, similar to [5]. TC uses a pairwise resolve function to compare records and a pairwise merge function to consolidate two matching records. It merges two records p_i and p_j as soon as resolve returns `true` to produce a new combined record $p_i \oplus p_j$, but it does not merge them when resolve returns `false`.

Let the order in which the TC algorithm invokes resolve functions be: $\Re(p_1, p_7) = \mathtt{t}$, $\Re(p_1 \oplus p_7, p_2) = \mathtt{f}$, $\Re(p_1 \oplus p_7, p_3) = \mathtt{f}$, $\Re(p_1 \oplus p_7, p_4) = \mathtt{f}$, $\Re(p_1 \oplus p_7, p_5) = \mathtt{f}$, $\Re(p_1 \oplus p_7, p_6) = \mathtt{f}$, $\Re(p_4, p_5) = \mathtt{f}$, $\Re(p_2, p_3) = \mathtt{t}$, $\Re(p_2 \oplus p_3, p_5) = \mathtt{f}$, $\Re(p_2 \oplus p_3, p_6) = \mathtt{f}$, $\Re(p_4, p_6) = \mathtt{f}$, $\Re(p_5, p_6) = \mathtt{t}$, $\Re(p_2 \oplus p_3, p_4) = \mathtt{t}$, $\Re(p_2 \oplus p_3 \oplus p_4, p_1 \oplus p_7) = \mathtt{f}$, $\Re(p_2 \oplus p_3 \oplus p_4, p_5 \oplus p_6) = \mathtt{f}$, $\Re(p_5 \oplus p_6, p_1 \oplus p_7) = \mathtt{f}$.

Above, "$\Re(p_i, p_j) = \mathtt{t/f}$" refers to the resolve function and its outcome (i.e., either `true` or `false`). Table 2 shows clustering $\mathcal{C} = \{C_1, C_2, C_3\}$ of relation R after applying TC. In $\mathcal{C}$ the duplicate publication records are merged into clusters, where the notation $p_i \oplus p_j$ denotes the merged representation of papers p_i and p_j. The clusters C_1, C_2, and C_3 have citation counts of 110 ($= 65 + 45$), 60 ($= 25 + 20 + 15$), and 15 ($= 10 + 5$), respectively[2].

In the above execution, 16 calls to the (potentially expensive) resolve function are made by TC. Query 1 on Table 2 returns the clusters C_1 and C_2 corresponding to $p_1 \oplus p_7$ and $p_2 \oplus p_3 \oplus p_4$, respectively.

Returned Answer Semantics. Before we illustrate how the knowledge of a query can be exploited to reduce the number of resolves invoked by the original merge algorithm, we first need to discuss the guarantees the QDA system can provide regarding the answer returned to the user. The system provides a trade-off between the strictness of the chosen query answer semantics and the efficiency of query processing, as explained below.

A QDA is said to follow *exact* semantics if the returned results match (in terms of both the clusters returned and their representations) the results returned by first cleaning the data (using TC) and then querying it (as in the standard solution above). For instance, when QDA follows *exact* semantics it must return $p_1 \oplus p_7$ (representing C_1) and $p_2 \oplus p_3 \oplus p_4$ (representing C_2) as the answer for Query 1. Note that such an answer will not have duplicates since each returned cluster will have a single representation.

An alternate answer might be $p_1 \oplus p_7$ (representing C_1) and $p_2 \oplus p_3$ (representing C_2). Note that the representation of C_2 is different (e.g., the value of *cited* is $25 + 20 = 45$, instead of 60). Here, the system does not provide the canonical merged representation as the TC algorithm; it does, however, return a semantically equivalent answer, since both C_1 and C_2 are represented in the returned answer. Yet again, each cluster is returned exactly once without duplication. In this case, QDA is said to follow *distinct* semantics.

Another possible returned answer, which provides more savings in terms of cleaning, is where duplicates are allowed to appear in the answer. For example, an answer such as: p_1, p_7, and $p_2 \oplus p_3$ clearly contains duplicates since p_1 and p_7 are returned as separate clusters although they represent C_1. This answer semantics is indeed consistent with the default `SQL` semantics of `SELECT` queries which return a *bag* instead of a *set* to prevent an expensive duplicate elimination. Similarly, in standard search (e.g., web, video, people, etc. search) end applications/users are already accustomed to tolerating duplicates in the answer. In this case, we say that QDA follows *representative* semantics.

Note that in all three semantics, the returned answer does not omit any cluster that is returned by the original ER algorithm. It also does not return any extra cluster that is not returned by the original merge algorithm. That is, the returned answer is always equivalent to an answer generated by the original TC algorithm. For instance, an answer such as: $p_1 \oplus p_7$ will not be acceptable since C_2 is not represented in the answer. Also, the answer $p_1 \oplus p_7$, $p_2 \oplus p_3$, and p_5 is not acceptable since p_5, which represents C_3, is returned while in fact C_3 does not satisfy the query.

As discussed above, the *exact* semantics is the most restrictive, followed by *distinct* semantics, then the *representative* semantics. Thus, and as shown next, the *representative* semantics provides the most opportunities for savings, followed by the *distinct* semantics, then the *exact* semantics.

QDA in Action. We next illustrate how QDA exploits query semantics to reduce the number of resolves. Assume *representative* semantics. Before any data cleaning step is invoked, we observe that both p_1 and p_7 have citation counts of 65 and 45, both of which satisfy the citation count criteria (i.e., ≥ 45), and thus should be present in the answer. As a result, depending on the order in which resolves are in-

[2]Assume that citations are split across different representations of the same paper and the cleaning algorithm sums them up to get the correct count.

voked, from 6 (e.g., $\Re(p_1, p_7)$ and $\Re(p_1 \oplus p_7, p_j)$) to 11 (e.g., $\Re(p_1, p_j)$, $\Re(p_7, p_j)$, and $\Re(p_1, p_7)$), where $j = 2, 3, 4, 5, 6$, calls (to resolve) can be eliminated. They are *vestigial*, as cluster C_1 will be represented in the answer set by both p_1 and p_7, regardless of the outcome of these calls. Furthermore, such resolves do not influence whether any additional clusters satisfy the query.

Suppose that QDA calls the next two resolves $\Re(p_4, p_5)$ and $\Re(p_2, p_3)$ with outcomes `f` and `t`, respectively, resulting in $p_2 \oplus p_3$ with citation count 45. Now, the query result would return answers p_1, p_7 and $p_2 \oplus p_3$ that represent clusters C_1 and C_2. Note that at this stage of the execution, all the remaining resolve function calls can be eliminated. The reason is that the remaining unresolved papers p_4, p_5, and p_6, whose citation counts are 15, 10, and 5, respectively, even if merged together, cannot form a new cluster C_3 distinct from C_1 and C_2, that satisfies the query predicate (since $15 + 10 + 5 = 30 < 45$). Thus, after only two calls to the resolve function, the QDA algorithm can safely and confidently return the answer p_1, p_7, and $p_2 \oplus p_3$, as all clusters matching the query have been found. Note that while the clusters returned by QDA and the original algorithm are exactly the same (viz., C_1 and C_2), their representations, $\{p_1 \oplus p_7, p_2 \oplus p_3 \oplus p_4\}$ versus $\{p_1, p_7, p_2 \oplus p_3\}$, are not the same. Also, note that the answer returned by QDA contains duplicates (p_1 and p_7), while the answer returned by TC does not since p_1 and p_7 are merged into cluster C_1.

To generate a *distinct* answer we can try a strategy to "isolate/disconnect" p_1 and p_7 from the remaining papers that would result in calls: $\Re(p_1, p_7) = \texttt{t}$, $\Re(p_1 \oplus p_7, p_2 \oplus p_3) = \texttt{f}$, $\Re(p_1 \oplus p_7, p_4) = \texttt{f}$, $\Re(p_1 \oplus p_7, p_5) = \texttt{f}$, $\Re(p_1 \oplus p_7, p_6) = \texttt{f}$ at which stage we can guarantee that $p_1 \oplus p_7$ and $p_2 \oplus p_3$ are distinct clusters in the answer of the original TC.

Now, to get the *exact* answer we need to add calls: $\Re(p_2 \oplus p_3, p_4) = \texttt{t}$, $\Re(p_2 \oplus p_3 \oplus p_4, p_5) = \texttt{f}$, $\Re(p_2 \oplus p_3 \oplus p_4, p_6) = \texttt{f}$.

Note that the original merge algorithm required 16 resolves, while QDA with *representative* semantics – 2 resolves, *distinct* semantics – 7 resolves, and *exact* semantics – 10 resolves, leading to savings in all three cases. In our experiments (Section 7), we will show that such a query-driven solution for all three semantics is significantly better compared to cleaning the whole dataset, especially when the query predicate is very selective.

4. NOTATION AND PROBLEM DEFINITION

We start this section by introducing common ER notation in Section 4.1. Then, we discuss new QDA-specific notation and formally define the problem in Section 4.2.

4.1 Standard Notation

Relation and Clustering. Let $R = \{r_1, r_2, \dots, r_{|R|}\}$ be a relation in the database, where r_k represents the k^{th} tuple of R and $|R|$ is its cardinality. Relation R is considered dirty if at least two of its records r_i and r_j represent the same real-world entity, and hence r_i and r_j are duplicates. The attributes in R can be represented as $\langle a_1, a_2, \dots, a_n\rangle$, where n is the arity of R. Thus, the k^{th} record in R is defined as $r_k = \langle \nu_{k1}, \nu_{k2}, \dots, \nu_{kn}\rangle$, where $\nu_{k\ell}$ is the value of the ℓ^{th} attribute in the k^{th} record (s.t. $1 \le k \le |R|$ and $1 \le \ell \le n$).

Recall that the goal of traditional ER is to partition records in R into a set of non-overlapping clusters $\mathcal{C} = \{C_1, \dots, C_{|\mathcal{C}|}\}$ such that each cluster corresponds to a single real-world entity. That is, any two records r_i and r_j from the same cluster

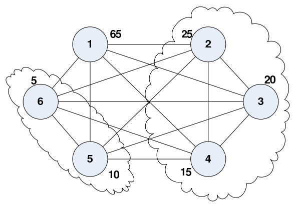

Figure 1: Graph G for R (Table 1) without p_7.

should co-refer, and simultaneously, any two records r_k and r_l from two distinct clusters C_m and C_n should not co-refer.

Graphical View of the Problem. The clustering problem can be represented graphically, as in [8,20], where records in R are encoded as a labeled graph $G = (V, E)$, where V is a set of nodes interconnected by a set of edges E. Each record $r_i \in R$ is represented by a node $v_i \in V$, hence $|V| = |R|$. Each edge $e_{ij} = (v_i, v_j)$ represents the possibility that r_i and r_j may be duplicates. In the simplest case, G is a complete graph with $|E| = \frac{|R|(|R|-1)}{2}$ edges. However, as we will explain in Section 5.3, our QDA approach will create a much simplified version of this graph. Figure 1 shows the encoding for R from Table 1 without p_7 for clarity. The numbers outside the nodes represent the *cited* count for each paper.

Resolve Function. A pairwise *resolve* function $\Re(r_i, r_j)$ operates on any two records $r_i, r_j \in R$ to try to decide whether they co-refer, that is, refer to the same real-world entity or not. Resolve is a "black-box" function that may be cheap or very expensive – e.g., a web query. The algorithms we develop are meant for the cases where the resolve function is *not* very cheap and calling resolves is in fact the bottleneck of an ER approach. The resolve function may return a classification, a binary answer, or a numeric similarity value (confidence). For the purpose of embedding resolve within an ER algorithm, the outcome of the resolve function is mapped into the following three decisions:

1. $\Re(r_i, r_j) = \texttt{MustMerge}$, if resolve is highly confident r_i and r_j are the same and hence, must be merged,
2. $\Re(r_i, r_j) = \texttt{MustSeparate}$, if resolve is highly confident r_i and r_j are different and hence, must be separated,
3. $\Re(r_i, r_j) = \texttt{Uncertain}$, otherwise.

By controlling when (i.e., for which similarity/dissimilarity levels) a resolve maps to each of these three decisions, the degree of eagerness can be controlled[3]. Naturally, the resolve may output decisions that are *incorrect* and that could lead to errors in the entity resolution process.

Merge and Combine Functions. If $\Re(r_i, r_j)$ returns `MustMerge`, then the two records are declared to be duplicates and a *merge* function $r_i \oplus r_j$ will consolidate them to produce a new record $r_m = r_i \oplus r_j$. To merge two duplicate records r_i and r_j, a *combine* function is used for each attribute a_ℓ s.t. $1 \le \ell \le n$.

We assume that the `WHERE`-attribute combine function $\nu_{i\ell} \oplus \nu_{j\ell}$ takes two values of attribute a_ℓ and outputs a single value $\nu_{m\ell} = \nu_{i\ell} \oplus \nu_{j\ell}$. Such combine functions perform different operations depending on the type of a_ℓ.

If a_ℓ is a numeric attribute then we consider:

- `ADD` semantics: $\nu_{i\ell} \oplus \nu_{j\ell} = \nu_{i\ell} + \nu_{j\ell}$,
- `MAX` semantics: $\nu_{i\ell} \oplus \nu_{j\ell} = \max(\nu_{i\ell}, \nu_{j\ell})$,

[3]For example, conceptually, R-Swoosh [5] merges `MustMerge` entities eagerly, but does not actually use `MustSeparate` – only `Uncertain`.

- MIN semantics: $\nu_{i\ell} \oplus \nu_{j\ell} = \min(\nu_{i\ell}, \nu_{j\ell})$.

The ADD semantics are used when records are obtained from the same data source, yet their entities are split in parts. The number of citations in duplicate publications in Google Scholar is an example of such a case. In this paper, ADD semantics are used as the default semantics to illustrate various examples, unless stated otherwise.

MAX semantics are used, for instance, when records are retrieved from different data sources where some copies of the record are obsolete. It is typically applied to attributes that monotonically increase over time, such as *age*. MIN semantics are similar to MAX, except that they are applied to attributes that monotonically decrease over time, such as *days to expire* attribute in a product table.

If a_ℓ is a categorical attribute then we consider:

- EXEMPLAR semantics: $\nu_{i\ell} \oplus \nu_{j\ell}$ chooses either $\nu_{i\ell}$ or $\nu_{j\ell}$ according to some policy,
- UNION semantics: $\nu_{i\ell} \oplus \nu_{j\ell} = \nu_{i\ell} \cup \nu_{j\ell}$.

The EXEMPLAR semantics are used when, for example, one value holds more information than the other value. For instance, in the *authors* attribute, the value "Alon Halevy" dominates "A. Halevy". In contrast, UNION semantics are utilized when the system needs to retain all possible values for some attribute, e.g., the application needs to retain all *email* addresses for an author.

Note that the afore-mentioned combine functions have the commutativity and associativity properties defined as:

1. Commutativity: $\nu_{i\ell} \oplus \nu_{j\ell} = \nu_{j\ell} \oplus \nu_{i\ell}$
2. Associativity: $(\nu_{i\ell} \oplus \nu_{j\ell}) \oplus \nu_{k\ell} = \nu_{i\ell} \oplus (\nu_{j\ell} \oplus \nu_{k\ell})$

Since these properties hold regardless of the merge order, the representation of the merged cluster will be the same.

4.2 Approach-Specific Notation

Queries. We will consider SQL selection queries. For clarity of presentation, our discussion will focus on queries with a single predicate p, with the syntax:

SELECT [DISTINCT|EXACT] $*$ FROM R WHERE a_ℓ *op* t

$$op \text{ is } \begin{cases} <, \leq, >, \geq, \text{ or } = & \text{if } a_\ell \text{ is a numeric attribute;} \\ = & \text{if } a_\ell \text{ is a categorical attribute.} \end{cases}$$

We will discuss the multi-predicate case in Section 5.

Returned Answer Equivalence. Before we formally define various answer semantics, we need to introduce several auxiliary concepts. Recall that a cluster C can be viewed as a set of records $C = \{r_1, r_2, \ldots, r_{|C|}\}$. We will say:

Definition 1. Record r_k *represents* cluster C, if $r_k \in C$, or $r_k = r_i \oplus r_j$ where r_i and r_j represent C.

Assume an entity resolution algorithm $\mathcal{A}$, such as TC, is applied to R and generates a clustering $\mathcal{C}_\mathcal{A}$ as its answer. $\mathcal{C}_\mathcal{A}$ is a set of clusters that partitions R. Let $\mathcal{C}_{\mathcal{A},Q}$ denotes the set of clusters from $\mathcal{C}_\mathcal{A}$ that satisfy query Q. Let $\mathcal{C}_{QDA,Q}$ be the set of clusters returned by QDA as the answer to Q.

To make our definitions formal, we also must account for the following observation: in general, the same algorithm $\mathcal{A}$ might produce a *different* clustering $\mathcal{C}'_\mathcal{A}$ of R, where $\mathcal{C}'_\mathcal{A} \neq \mathcal{C}_\mathcal{A}$, if $\mathcal{A}$ changes the *order* in which it invokes resolves [27]. Let $\bar{\mathcal{C}}_\mathcal{A}$ denotes the set of all possible output clusterings that $\mathcal{A}$ may produce as a result of changing its resolves order.

Now we can define when $\mathcal{C}_{QDA,Q}$ is exactly, distinctly, or representationally equivalent to an answer of $\mathcal{A}$ to query Q:

Definition 2. Answer $\mathcal{C}_{QDA,Q}$ generated by QDA for query Q is *exactly* equivalent to that of algorithm $\mathcal{A}$ iff there exists $\mathcal{C}_\mathcal{A} \in \bar{\mathcal{C}}_\mathcal{A}$ such that: (a) for each cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ there exists *exactly one* cluster $C_j \in \mathcal{C}_{QDA,Q}$ such that $C_i \equiv C_j$, and (b) for each cluster $C_j \in \mathcal{C}_{QDA,Q}$ there exists *exactly one* cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ such that $C_j \equiv C_i$.

In other words, there is a one-to-one mapping between clusters in $\mathcal{C}_{QDA,Q}$ and $\mathcal{C}_{\mathcal{A},Q}$ and the content of clusters in $\mathcal{C}_{QDA,Q}$ is *identical* to those in $\mathcal{C}_{\mathcal{A},Q}$.

We further define the less restrictive *distinct* semantics as:

Definition 3. Answer $\mathcal{C}_{QDA,Q}$ generated by QDA for query Q is *distinctly* equivalent to that of algorithm $\mathcal{A}$ iff there exists $\mathcal{C}_\mathcal{A} \in \bar{\mathcal{C}}_\mathcal{A}$ such that: (a) for each cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ there exists *exactly one* cluster $C_j \in \mathcal{C}_{QDA,Q}$ such that $C_i \supseteq C_j$, and (b) for each cluster $C_j \in \mathcal{C}_{QDA,Q}$ there exists *exactly one* cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ such that $C_j \subseteq C_i$.

That is, there is still a one-to-one mapping between clusters in $\mathcal{C}_{QDA,Q}$ and $\mathcal{C}_{\mathcal{A},Q}$, but now clusters in $\mathcal{C}_{QDA,Q}$ are allowed to be *subsets* of clusters from $\mathcal{C}_{\mathcal{A},Q}$.

We define the least restrictive *representative* semantics as:

Definition 4. Answer $\mathcal{C}_{QDA,Q}$ generated by QDA for query Q is *representationally* equivalent to that of algorithm $\mathcal{A}$ iff there exists $\mathcal{C}_\mathcal{A} \in \bar{\mathcal{C}}_\mathcal{A}$ such that: (a) for each cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ there exists *at least one* cluster $C_j \in \mathcal{C}_{QDA,Q}$ such that $C_i \supseteq C_j$, and (b) for each cluster $C_j \in \mathcal{C}_{QDA,Q}$ there exists *exactly one* cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ such that $C_j \subseteq C_i$.

The *representative* semantics goes one step further on top of the distinct semantics and does not require the one-to-one mapping by allowing for duplicates. Namely, it asks for one-to-many mapping from $\mathcal{C}_{\mathcal{A},Q}$ to $\mathcal{C}_{QDA,Q}$ and one-to-one mapping from $\mathcal{C}_{QDA,Q}$ to $\mathcal{C}_{\mathcal{A},Q}$.

Problem Definition. Let $\mathcal{A}$ be the original entity resolution algorithm, e.g., TC, whose query-driven version is being developed. Then, given a query Q, we can formally define our problem as an optimization problem as follows:

Minimize: Number of $\Re()$
Subject to:
1. $\forall C \in \mathcal{C}_{QDA,Q}$, C satisfies Q; // *Query satisfaction*
2. $\mathcal{C}_{QDA,Q} \equiv \mathcal{C}_{\mathcal{A},Q}$; // *User-defined equivalence*

It can be trivially shown that achieving an optimal solution that generates the least number of resolves is *infeasible* in practice, as it requires an "oracle" that knows which pair of records to resolve next. Thus, the goal translates into finding a *good* solution by developing algorithms which attempt to reduce the number of calls to the resolve function by exploiting vestigiality, and by implementing a good edge selection policy, as explained in Section 6.

5. VESTIGIALITY

In this section, we introduce the notion of *vestigiality*, which is the key concept in our query-driven solution. Before we can formally define it, we have to introduce several auxiliary concepts. We first define a way to categorize a triple $(p, \oplus, a_\ell)$ (where p is the query predicate, $\oplus$ is the combine function defined over a_ℓ's domain) into three categories: *in-preserving*, *out-preserving*, and *neither* as explained in Section 5.1. Then, we discuss how to deal with multi-predicate selection queries in Section 5.2. The construction of the labeled graph is explained in Section 5.3. Finally, we explain how this categorization as well as the new notions of *relevant clique* and *minimal clique* can be used to test for vestigiality of an edge in Section 5.4.

$\oplus$, domain	$a_\ell \geq t$ or, $a_\ell > t$	$a_\ell \leq t$ or, $a_\ell < t$	$t_1 \leq a_\ell \leq t_2$
`ADD`, $a_\ell \in \mathbb{R}^+$	in-preserving	out-preserving	neither
`MAX`, $a_\ell \in \mathbb{R}$	in-preserving	out-preserving	neither
`MIN`, $a_\ell \in \mathbb{R}$	out-preserving	in-preserving	neither

Table 3: Triple categorization for numerics.

$\oplus$, domain	$a_\ell = t$
`EXEMPLAR`, $a_\ell \in$ enum	in-preserving
`UNION`, $a_\ell \in$ enum	in-preserving

Table 4: Triple categorization for categoricals.

$\tau_i = (p_i, \oplus_i, a_i)$	$\tau_j = (p_j, \oplus_j, a_j)$	$\tau_i \wedge \tau_j$	$\tau_i \vee \tau_j$
in-preserving	in-preserving	in-preserving	in-preserving
in-preserving	out-preserving	neither	neither
out-preserving	out-preserving	out-preserving	out-preserving
in-preserving	neither	neither	neither
out-preserving	neither	neither	neither
neither	neither	neither	neither

Table 5: Triples generalization

5.1 Triple $(p, \oplus, a_\ell)$ Categorization

QDA exploits the specificity of a query predicate p and the semantics of a combine function $\oplus$ defined on attribute a_ℓ to significantly reduce the cleaning overhead by resolving *only* those edges that may influence the answer of Q. For that goal, we will classify any triple $(p, \oplus, a_\ell)$ into three generic categories: *in-preserving*, *out-preserving*, and *neither*. These broad categories are important as they allow us to develop generic QDA algorithms instead of developing specific algorithms for each small case.

Definition 5. Triple $(p, \oplus, a_\ell)$ is *in-preserving*, if for all possible values $\nu_{i\ell}, \nu_{j\ell} \in a_\ell$, if p is `true` for $\nu_{i\ell}$, then p is also `true` for all $\nu_{i\ell} \oplus \nu_{j\ell}$.

This property means that once a record is `in` the answer, it will remain so, even if it is merged with other records. For instance, $(\textit{cited} \geq 45, \texttt{ADD}, \textit{cited})$ is in-preserving, since any tuple with a citation count above or equal to 45 will continue to be above or equal to 45 even if merged with other tuples. In contrast, $(\textit{cited} \leq 45, \texttt{ADD}, \textit{cited})$ is not in-preserving.

Definition 6. Triple $(p, \oplus, a_\ell)$ is *out-preserving*, if for all possible values $\nu_{i\ell}, \nu_{j\ell} \in a_\ell$, if p is `false` for $\nu_{i\ell}$, then it is also `false` for all $\nu_{i\ell} \oplus \nu_{j\ell}$.

This property means that once a record is `out` of the answer, it will remain so, even if it is merged with other records. E.g., $(\textit{cited} \leq 45, \texttt{ADD}, \textit{cited})$ is out-preserving.

Tables 3 and 4 show a classification of different common triples for numeric and categorical attributes, respectively.

5.2 Multi-Predicate Selection Queries

Our discussion so far has focused on the case where the `WHERE`-clause contains a single predicate. The overall solution, however, applies to more complex selection queries with multiple predicates connected via logical connectives, such as `AND`, `OR`, and `NOT`. This is since such combinations of triples can also be categorized into the same three categories – based on the categories of the basic triples it is composed of, as illustrated in Table 5, see [1] for proofs. For instance, consider the following *range* query:

Query 2. `SELECT` $*$ `FROM` R `WHERE` *cited* $\geq$ 45 `AND` *cited* ≤ 65

This range query consists of two basic predicates p_1 : *cited* ≥ 45 and p_2 : *cited* ≤ 65. Hence, it consists of two

```
CREATE-GRAPH($R, Q, A_{cur}, V_{out}, V_{maybe}, \oplus$)
1  for each $r_k \in R$ do
2      $v_k \leftarrow$ CREATE-NODE($r_k$)
3      if IS-IN-PRESERVING($p, \oplus, a_\ell$) and SATISFY-QRY($v_k, Q$) then
4          $A_{cur} \leftarrow A_{cur} \cup \{v_k\}$
5      else if IS-OUT-PRESERVING($p, \oplus, a_\ell$)
           and not SATISFY-QRY($v_k, Q$) then
6          $V_{out} \leftarrow V_{out} \cup \{v_k\}$
7      else $V_{maybe} \leftarrow V_{maybe} \cup \{v_k\}$
8  $V \leftarrow \{A_{cur}, V_{out}, V_{maybe}\}$
9  $E \leftarrow$ CREATE-EDGES-WITH-BLOCKING($V_{maybe}, V_{maybe}$)
10 $E \leftarrow E \cup$ CREATE-EDGES-WITH-BLOCKING($V_{out}, V_{maybe}$)
11 return $G(V, E)$
```

Figure 2: Create graph function.

triples: an in-preserving triple $\tau_1 = (\textit{cited} \geq 45, \texttt{ADD}, \textit{cited})$ and an out-preserving $\tau_2 = (\textit{cited} \leq 65, \texttt{ADD}, \textit{cited})$. From Table 5, we can see that the resulting combination $\tau_1 \wedge \tau_2$ is neither in- nor out-preserving. To see why, consider record r_i with *cited* $= 40$. Initially r_i is `out` of the answer of Query 2. If r_i merges with another record r_j with *cited* $= 10$, the new record $r_i \oplus r_j$ will have *cited* $= 40 + 10 = 50$ and will be `in` the answer. If now $r_i \oplus r_j$ can merge again with another record r_k with *cited* $= 20$, then $r_i \oplus r_j \oplus r_k$ will have *cited* $= 50 + 20 = 70$ which clearly does not satisfy Query 2. Hence, $\tau_1 \wedge \tau_2$ is neither in-preserving nor out-preserving.

5.3 Creating and Labeling the Graph

To formally define vestigiality testing, we need to explain how QDA builds and labels the graph, see CREATE-GRAPH() function in Figure 2. The main goal of this function is to avoid creating as many nodes and edges as possible in order to improve the efficiency. As common for ER techniques, the function starts by applying blocking and will not create edges for pairs that cannot be duplicates according to blocking. More importantly, on top of blocking, the function will also remove from consideration nodes and edges that will not influence further processing of Q, thus improving the efficiency on top of blocking from the very beginning.

The algorithm starts by iterating over each tuple $r_k \in R$ to create the corresponding node v_k. It sets label $\ell[v_k]$ of v_k (Lines 1–7) as:

1. $\ell[v_k] =$ `in` when triple $(p, \oplus, a_\ell)$ is in-preserving and v_k satisfies Q. Node v_k is added to A_{cur} as it is guaranteed to be in the final answer.
2. $\ell[v_k] =$ `out` when triple $(p, \oplus, a_\ell)$ is out-preserving and v_k does not satisfy Q. Node v_k is added to V_{out}.
3. $\ell[v_k] =$ `maybe`, otherwise. Node v_k is added to V_{maybe}.

The algorithm then creates edges, but only if they can exist according to blocking and (1) only among nodes in V_{maybe} and (2) for each v_i, v_j pair where $v_i \in V_{out}$ and $v_j \in V_{maybe}$ (Lines 9–10). This is because nodes in V_{maybe} that merge with V_{out} nodes cannot be in the answer. For each edge $e_{ij} \in E$, QDA sets labels as:

1. $\ell[e_{ij}] =$ `yes`, when $\Re(r_i, r_j)$ has already been called and returned `MustMerge`,
2. $\ell[e_{ij}] =$ `no`, when $\Re(r_i, r_j)$ has already been called and returned `MustSeparate`,
3. $\ell[e_{ij}] =$ `maybe`, when $\Re(r_i, r_j)$ has already been called and returned `Uncertain`,
4. $\ell[e_{ij}] =$ `vestigial`, when, Definition 8 holds. Note that as QDA proceeds forward, some edges that were not vestigial previously may become vestigial. But once they become vestigial, they remain so,

5. $\ell[e_{ij}] =$ `unresolved`, otherwise.

It should be noted that edge labeling is a convenient semantic notation useful for explaining various concepts. For efficiency, however, the algorithm does not utilize `yes` and `no` labels in its actual processing. For example, instead of labeling edge e_{ij} as a `no` edge, it simply removes this edge since this simplifies the graph. Similarly, instead of labeling an edge $e_{ij} = (v_i, v_j)$ as a `yes` edge, the algorithm merges nodes v_i and v_j into a new node $v_m = v_i \oplus v_j$. We will say that the current labeling of the graph determines the *current state* of the resolution process. Now we can define the concept of the current answer A_{cur}.

Definition 7. Based on the given edge labeling, the *current answer* A_{cur} to Q is the answer resulting from assuming that all `vestigial` and `unresolved` edges are `no` edges.

Example 1. Consider relation R in Figure 1 and an in-preserving triple e.g., (*cited* ≥ 45, `ADD`, *cited*). Initially only p_1 is labeled `in`, since it is guaranteed to be in the result-set ($A_{cur} = \{p_1\}$). Thus, all edges incident to p_1, that is e_{12}, e_{13}, e_{14}, e_{15}, and e_{16}, are `vestigial`. If the algorithm then calls $\Re(p_2, p_3)$ and $\Re(p_4, p_5)$, the corresponding edges will be assigned labels $\ell[e_{23}] =$ `yes` and $\ell[e_{45}] =$ `no`.

5.4 Vestigiality Testing Using Cliques

Before introducing the new notions of relevant/minimal cliques which are used to test for vestigiality of an edge, let us first define the concept of a `vestigial` edge. Intuitively, an edge is vestigial if its resolution outcome does not influence the query result. Formally:

Definition 8. Let $\mathcal{A}$ be the original entity resolution algorithm. An edge $e_{ij} \in E$ is `vestigial` when, regardless of what the ground truth for e_{ij} might be, QDA can guarantee that by treating e_{ij} as a `no` edge, it can still compute an equivalent answer to that of $\mathcal{A}$.

Now, we introduce some of the necessary concepts utilized in our solution. We use the standard definition of a *clique* in an undirected graph $G = (V, E)$, which is a subset of the node set $S \subseteq V$, such that for every two nodes in S, there exists an edge in E that connects them. A clique is an important concept for entity resolution since it identifies which groups of nodes/records might or might not co-refer:

Lemma 1. Nodes (records) co-refer only if they form a clique consisting of only `yes` edges in the ground truth.

Consequently, if a group of nodes is not a clique (e.g., some edges are marked `no` (i.e., removed)), and the algorithm did not make a mistake in removing those edges, then that group corresponds to at least two distinct entities. Note that Lemma 1 deals with the ground truth labels and not the decisions returned by the resolve function.

Let $\mathcal{C}_{cur}$ be the set of clusters in the current answer A_{cur}. Now, we can define the notions of a relevant clique and a minimal clique.

Definition 9. A clique S is called *relevant* to Q, if we can assign labels to its edges such that this labeling might change $\mathcal{C}_{cur}$, by either adding (at least one) new cluster to $\mathcal{C}_{cur}$, or removing (at least one) cluster from $\mathcal{C}_{cur}$.

The concept of relevant cliques provides a mechanism to test if an edge is vestigial as stated in the next theorem.

```
IS-VESTIGIAL($e_{ij}, G, Q, \oplus$)
1  if IS-IN-PRESERVING($p, \oplus, a_\ell$) then
2      return not IS-IN-A-MINIMAL-CLIQUE($e_{ij}, G, Q$)
3  else return not IS-IN-A-RELEVANT-CLIQUE($e_{ij}, G, Q$)
```

Figure 3: Is vestigial function

Theorem 1. Given the current labeled graph G, a selection query Q with predicate p on attribute a_ℓ, if no relevant clique exists that includes e_{ij}, then e_{ij} is vestigial. However, the reverse does not hold: a vestigial edge could be part of a relevant clique. Proof is covered in [1].

The next example helps in illustrating the importance of considering relevant cliques.

Example 2. Consider G shown in Figure 1 after resolving e_{23} only and Query 1, $p =$ (*cited* ≥ 45). $A_{cur} = \{p_1, p_2 \oplus p_3\}$ because both p_1 and $p_2 \oplus p_3$ have citation counts ≥ 45 (65 and 45, respectively). As a result, all edges incident to p_1 and $p_2 \oplus p_3$ are vestigial. Note that nodes 4, 5, and 6 form a clique S. The sum up of the *cited* attribute for these nodes is $15 + 10 + 5 = 30 \not\geq 45$. Thus, merging nodes in S cannot change A_{cur} (no new clusters can be added to it). Hence, S is not a relevant clique w.r.t. p. Thus, edges e_{45}, e_{46}, and e_{56} are not part of any relevant cliques and hence, vestigial.

In fact, when $(p, \oplus, a_\ell)$ is in-preserving, we can show that the edge must not only be part of a relevant clique, but a minimal clique as defined below:

Definition 10. A relevant clique S is called a *minimal clique*, if no subset of nodes in S can form a relevant clique.

Theorem 2. Given a graph G and an in-preserving $(p, \oplus, a_\ell)$, an unresolved edge e_{ij} is vestigial if and only if no minimal clique exists that includes e_{ij}. Proof is covered in [1].

The next example shows the concept of minimal cliques.

Example 3. Consider G shown in Figure 1 after resolving e_{45} only. Note that the triple (*cited* ≥ 45, `ADD`, *cited*) is in-preserving. Observe that $S = \{p_2, p_3, p_4\}$ is a relevant clique ($25 + 20 + 15 = 60 \geq 45$). However, there exists $S_{min} = \{p_2, p_3\} \subset S$ which forms a *minimal* clique ($25 + 20 = 45 \geq 45$). Therefore, edges e_{24} and e_{34} are vestigial since both e_{24} and e_{34} do not belong to any minimal clique.

The above two theorems suggest that testing for vestigiality can be implemented by checking for relevant/minimal cliques as shown in IS-VESTIGIAL() function, see Figure 3. However, finding such cliques is NP-hard as shown in the next theorem.

Theorem 3. Testing for vestigiality using IS-VESTIGIAL() is NP-hard. This can be shown through a straightforward reduction from the well-known k-clique problem, and hence is computationally infeasible. Full proof is covered in [1].

Thus, implementing IS-VESTIGIAL() is impractical as the naive algorithm that calls all the $O(n^2)$ resolves is going to be faster. Consequently, the challenge is to design a QDA strategy that still performs vestigiality testing, but does it fast enough to outperform the naive approach. Thus, in the next section, we will explain how to devise efficient approximation-based techniques for vestigiality testing.

6. QUERY-DRIVEN SOLUTION

In this section we describe our QDA approach. We begin by presenting an overview of the framework. Next, we explain the framework components in more detail.

6.1 Overview of The Approach

The main task of the QDA approach is to compute an answer to query Q very efficiently. The answer should be equivalent to first applying a standard algorithm, such as transitive closure (TC) on the whole dataset and then querying the resulting cleaned data with query Q.

Recall that traditional TC operates by iteratively choosing a pair of nodes to resolve next, then applying the resolve function, merging nodes if the resolve returns a positive answer, and then repeating the process. The QDA approach is very similar to the original TC with two noticeable differences. First, QDA uses its own pair-picking strategy to select pairs of nodes to resolve next. The goal of this strategy is to minimize the number of calls to resolve to answer the given query. Second, instead of calling resolve on the chosen pair, QDA first tries to quickly determine if it can avoid making this call altogether by checking if the chosen pair is vestigial.

Conceptually, the QDA approach can be viewed as consisting of the following steps:

1. *Creating and Labeling the Graph.* The approach starts by creating and labeling graph G (Section 5.3).
2. *Choosing an Edge to Resolve.* Based on its edge-picking policy, the approach selects edge e_{ij} to resolve. Intuitively, such a policy should select e_{ij} in a way that resolving it would allow QDA to either quickly *add* some cluster-representative to the result-set, or would *break* many relevant cliques. We have experimented with many different policies. The one that has demonstrated the best results is based on picking edges according to their weight, where weight w_{ij} for edge e_{ij} is computed by combining the values of its incident nodes: $w_{ij} = \nu_{i\ell} \oplus \nu_{j\ell}$. The edge-picking policy is not our focus in this paper.
3. *Lazy Edge Removal.* We have implemented many optimizations in QDA, here we briefly describe one of them. In this step the algorithm checks if the chosen edge e_{ij} still exists. If it does not, then the algorithm will go back to Step 2 to pick another edge. Note that e_{ij} can disappear as the result of merging of two nodes v_k and v_l. Observe that after merging v_k and v_l, only edges that are common to both of them must remain in G. But checking for common edges and then aggressively removing them from auxiliary data structures at the time of the merge is an $O(|R|)$ operation in general for each merge operation. To reduce this cost, QDA does not remove the edges at the time of the merge, but removes them *lazily* in this step. It does so in $O(1)$ time by checking if v_i (or v_j) of edge e_{ij} has been merged with some other node v_k by the algorithm on a prior iteration, and hence (1) v_i (or v_j) was removed from V_{maybe}, or (2) v_i is not in v_j's neighborhood or vice versa.
4. *Vestigiality Testing.* The algorithm, in this step, tries to avoid calling resolve on edge e_{ij} by checking if it is `vestigial` (Section 6.2).
5. *Stopping Condition.* If there exists an edge $e_{ij} \in E$ that is neither `resolved` nor `vestigial`, then the algorithm iterates by going to Step 2.
6. *Computing the Answer.* Finally, the algorithm computes the query's final answer using the required answer semantics S (Section 6.3).

Thus, our goal translates into designing algorithms that implement the above steps. Such algorithms should minimize the number of invocations of the expensive resolve function, and be able to correctly and efficiently find an answer to a given query. In addition, the chosen algorithms *must themselves be very efficient*, otherwise their cost will dominate the cost of calling the resolve function, and a simple strategy such as resolving all $O(n^2)$ edges in random order might be more efficient.

The following sections elaborate all of these steps in detail.

```
VESTIGIALITY-TESTING($e_{ij}, G, Q, \oplus$)
1  if IS-IN-PRESERVING($p, \oplus, a_\ell$)
   and MIGHT-CHANGE-ANSWER($\emptyset, v_i \oplus v_j, Q$) then
2      $res \leftarrow \Re(v_i, v_j)$
3      if $res$ = MustMerge then
4          $A_{cur} \leftarrow A_{cur} \cup \{v_i \oplus v_j\}$
5          $V_{maybe} \leftarrow V_{maybe} - \{v_i, v_j\}$
6      else if $res$ = MustSeparate then
7          $E \leftarrow E - \{e_{ij}\}$
8      else $\ell[e_{ij}]$ = maybe
9  else if CHECK-POTENTIAL-CLIQUE($e_{ij}, G, Q$) then
10     $res \leftarrow \Re(v_i, v_j)$
11     if $res$ = MustMerge then
12         $v_i \leftarrow v_i \oplus v_j$
13         $\mathcal{N}[v_i] = \mathcal{N}[v_i] \cap \mathcal{N}[v_j]$
14         $V_{maybe} \leftarrow V_{maybe} - \{v_j\}$
15     else if $res$ = MustSeparate then
16         $E \leftarrow E - \{e_{ij}\}$
17     else $\ell[e_{ij}]$ = maybe
18 else $E \leftarrow E - \{e_{ij}\}$     // this edge is vestigial
```

Figure 4: Vestigiality testing function

6.2 Vestigiality Testing

Given an edge e_{ij} selected by the edge-picking strategy, the main task of vestigiality testing is to determine if e_{ij} is vestigial and thus calling resolve on it can be avoided. However, from Section 5, we know that testing for the exact vestigiality via clique-checking is an NP-hard problem. Hence, QDA employs a highly efficient approximate solution instead of the exact check. Namely, QDA tests for vestigiality by using an inexact-but-fast check to determine if e_{ij} can *potentially* be part of any relevant clique at all.

The vestigiality testing function, provided in Figure 4, can conceptually be viewed as consisting of the following steps:

1. *Edge Minclique Check Optimization.* This is another optimization which the algorithm employs (Lines 1–8). Here, a check for a special case allows the algorithm to remove two nodes from the graph instead of one in case of a merge, leading to extra savings. Namely, this special case exists when triple $(p, \oplus, a_\ell)$ is in-preserving and edge e_{ij} by itself can change the current answer to Q. If so, then e_{ij} is not vestigial and the algorithm calls resolve on it. Now, if resolve returns `MustMerge`, then the algorithm adds the merged node $(v_i \oplus v_j)$ to the answer set and then removes v_i and v_j nodes from G. The algorithm can perform this optimization because v_i and v_j are already represented by their merged representation in A_{cur}.
2. *Check for Potential Clique.* If Step 1 does not apply, then the function calls the CHECK-POTENTIAL-CLIQUE() procedure to test if e_{ij} can *potentially* be part of any relevant clique at all. If the call returns `true`, then the function calls resolve function on e_{ij} and labels G accordingly (Lines 10–17). If the call returns `false`, then the function marks e_{ij} as `vestigial` (Line 18).

The key intuition behind the CHECK-POTENTIAL-CLIQUE() function is to *quickly* check if an edge e_{ij} can *potentially* be

```
CHECK-POTENTIAL-CLIQUE($e_{ij}, G, Q$)
 1  $v_{new} \leftarrow v_i \oplus v_j$
 2  if MIGHT-CHANGE-ANSWER($\emptyset, v_{new}, Q$) then
 3      return true
 4  $V_{intersect} \leftarrow \mathcal{N}[v_i] \cap \mathcal{N}[v_j]$
 5  for each $v_k \in V_{intersect}$ do
 6      $v_{old} \leftarrow v_{new}$
 7      $v_{new} \leftarrow v_{old} \oplus v_k$
 8      if MIGHT-CHANGE-ANSWER($v_{old}, v_{new}, Q$) then
 9          return true
10  return false
```

Figure 5: Check potential clique function

```
COMPUTE-ANSWER($A_{cur}, V_{maybe}, Q, S$)
 1  for each $v_i \in V_{maybe}$ do
 2      if SATISFY-QRY($v_i, Q$) then
 3          $A_{cur} \leftarrow A_{cur} \cup v_i$
 4  if $S =$ Distinct then
 5      for each $v_i \in A_{cur}$ do
 6          for each $v_j \neq v_i \in A_{cur}$ do
 7              if $\Re(v_i, v_j) =$ MustMerge then
 8                  $v_i \leftarrow v_i \oplus v_j$
 9                  $A_{cur} \leftarrow A_{cur} - \{v_j\}$
10  else if $S =$ Exact then
11      for each $v_i \in A_{cur}$ do
12          for each $v_j \neq v_i \in A_{cur} \cup V_{maybe}$ do
13              if $\Re(v_i, v_j) =$ MustMerge then
14                  $v_i \leftarrow v_i \oplus v_j$
15                  if $v_j \in A_{cur}$ then
16                      $A_{cur} \leftarrow A_{cur} - \{v_j\}$
```

Figure 6: Compute answer function

involved in a relevant/minimal clique at all, see Figure 5. It is a *safe* approximate function: it returns `false` *only* when e_{ij} is guaranteed not to be a part of any relevant/minimal clique (i.e., vestigial). In contrast, it returns `true` when e_{ij} *might* be a part of some relevant clique and thus, the algorithm cannot guarantee it is vestigial. For efficiency, the algorithm treats it as non-vestigial without performing any further costly checks.

The idea of this function can be illustrated with the following example. Assume two nodes v_i and v_j with citation counts of 10 and 10, and further assume that they have only two common neighbors whose citation counts are 15 and 20. Then edge e_{ij} cannot be part of any clique with combined citation count above $10 + 10 + 15 + 20 = 55$. Also note that while this (v_i, v_j) edge *might* be a part of a clique with *cited* $= 55$, the algorithm cannot guarantee that without checking the existence of the edge between 15 and 20.

The function that utilizes this intuition is illustrated in Figure 5. It begins by merging nodes v_i and v_j and then checking if their merge might change Q's answer. If it does not, then the function computes the intersection of v_i and v_j neighborhoods (Line 4) and then, tries to find the *smallest* potential clique from their common neighbors which might change Q's answer. The function will keep expanding the size of such clique until no common neighbors are left. Once the function succeeds in finding a potential clique that might change Q's answer then it will return `true` (Lines 5-9). Otherwise, it returns `false` (Line 10).

6.3 Computing Answer of Given Semantics

After the algorithm is done processing edges, it computes its final answer A_{cur} to query Q based on the answer semantics S the user requested. For that, it uses the COMPUTE-ANSWER() function illustrated in Figure 6. The function starts by adding nodes from V_{maybe} which satisfy Q to A_{cur} (Lines 1–3). At this stage A_{cur} satisfies *representative* answer semantics. As such, A_{cur} might contain duplicates and/or it might not be equivalent to the canonical merged representation produced by the original TC algorithm.

If the user requests stricter *distinct* or *exact* answer semantics, then the algorithm continues building the corresponding answers based on the current A_{cur}. The algorithm implements *distinct* semantics by cleaning the (small) representative answers in A_{cur} by using the original TC algorithm. That is, duplicates are removed by resolving all pairs of nodes in A_{cur} (Lines 4–9). Thus, the additional cost of cleaning this small result-set is $O(|A_{cur}|^2)$ resolves in the worst case, where $|A_{cur}|$ is the size of the result-set.

To generate an answer that satisfies *exact* semantics, the algorithm proceeds by comparing clusters in the result-set with clusters that are not in the result-set. That is, it compares all nodes in A_{cur} with all nodes in $A_{cur} \cup V_{maybe}$ (Lines 10–16). Therefore, the extra cost of cleaning leads to $O(|A_{cur}||R|)$ additional resolves in the worst case.

Note that to produce *distinct* or *exact* answer, edges previously labeled `vestigial` are considered `unresolved` edges.

6.4 Answer Correctness

From a theoretical perspective, it could be useful to analyze the properties of our QDA algorithm with respect to answer correctness. Note that if the resolve function is always accurate, then TC will compute clustering $\mathcal{C}$ that is identical to the ground-truth clustering $\mathcal{C}_{gt}$. Consequently, the following lemma holds trivially:

Lemma 2. If the resolve function is always accurate, then QDA will compute answers that are: *representationally*, *distinctly*, or *exactly* equivalent to those in $\mathcal{C}_{gt}$.

Lemma 2 essentially states a theoretical result: the QDA algorithm is guaranteed to compute the correct answer, provided that the resolve function is accurate. Naturally, resolve functions are not always accurate, and hence no ER technique can guarantee the correctness of its answer. We also do not assume that resolve is always accurate.

6.5 Discussion

In this paper we have laid out the foundations of the generic query-driven entity resolution framework. While we have considered a broad class of `SQL` selection queries, we have not yet considered all `SQL` queries, e.g., joins. The latter are future directions of our work. We have covered very generic algorithms that apply to a broad class of cases that are based on categorizing triples into in-/out-preserving and neither. However, various optimizations of these algorithms are possible when considering each specific case separately. Namely, we have studied different implementations of equality and range style queries in [1]. The idea is to use two-stage processing. In the first stage an edge is resolved only if it is part of a relevant clique and thus it can result in a new node that should be in the answer. Thus, all nodes that must be in the answer will be in A_{cur}, but A_{cur} is a superset of nodes, as it may contain *wrong* nodes that should not be there. To remove the wrong nodes from A_{cur}, the second stage of the algorithm resolves edges only if an edge is a part of a clique that includes at least one node in A_{cur} and that clique can change the answer for that node from being `in` A_{cur} to being `out` of A_{cur}. This two-stage strategy leads to a noticeable

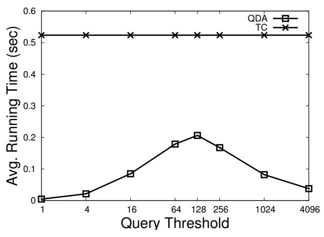

Figure 7: QDA vs. TC [Time]

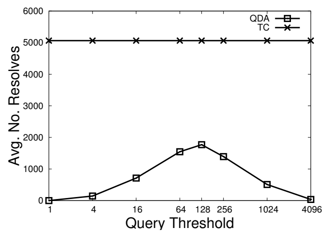

Figure 8: QDA vs. TC [$\#\Re()$]

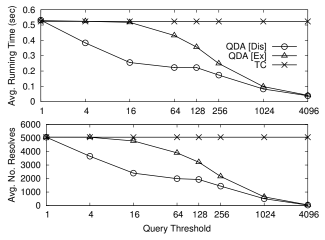

Figure 9: Answer semantics

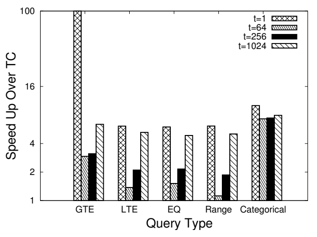

Figure 10: QDA speed up

improvement in processing range queries. Also note that, in this paper, we have presented QDA in the context of transitive closure-like algorithms, but similar techniques apply to some other types of clustering algorithms. Specifically, we show how to develop QDA for the very complex case of correlation clustering in [1].

7. EXPERIMENTAL EVALUATION

In this section we empirically evaluate the efficiency of our QDA approach on real data. We study QDA for different query types (GTE, LTE, etc.) and compare it to TC in terms of, both, the end-to-end running time and the number of calls to resolve. The experiments will show how the chosen answer semantics affect the efficiency of QDA. We also study the impact of vestigiality testing as the cost of resolve functions is varied. We will demonstrate the result of using/not-using blocking on both QDA and TC. Finally, we analyze the effectiveness of our greedy edge-picking strategy.

7.1 Experimental Setup

Google Scholar Dataset. We ran our tests on a real bibliographic dataset collected from Google Scholar. The dataset represents publications of the top 50 computer science researchers each having h-index of 60 or higher [3]. The dataset schema is similar to that of Table 1. The dataset consists of $16,396$ records where 14.3% are duplicates.

Resolve Function. We have implemented a highly-accurate pairwise resolve function which operates on two records $r_i, r_j \in R$ to decide whether they refer to the same real-world entity. The resolve function utilizes Soft-TF-IDF [10] to compare titles and the Jaro-Winkler distance to compare author names. If the similarity is sufficient (above threshold) and there are $min(|A_i|, |A_j|)$ authors in common ($|A_i|$ and $|A_j|$ are the number of authors), then resolve returns `MustMerge` and r_i, r_j are considered to be duplicates.

Blocking Technique. Both TC and QDA use the same blocking procedure. Namely, we use two blocking functions to cluster records that might be duplicates together. The first function partitions records (i.e., papers) into buckets based on the first two letters of their titles. Similarly, the second one partitions them based on the last two letters. That is, if two papers match in either their first two letters or their last two letters then they are put in the same block.

7.2 Experiments

Experiment 1 (QDA vs. TC). Figures 7 to 9 use a set of GTE ($\geq$) queries to show the effects of vestigially testing by comparing our QDA algorithm (using representative, distinct, and exact answer semantics) with TC. These algorithms are compared in terms of their end-to-end running time and the number of resolves called.

Figure 7 plots the actual end-to-end execution time of QDA (for representative answer semantics) and TC for different values of the GTE query threshold t. Figure 8 is similar but plots the number of calls to resolve instead of the execution time. These are log-lin scale plots, where t's range is chosen to show the entire spectrum of QDA's behavior. Note that the curves in Figures 7 and 8 are similar, thus demonstrating the fact that the calls to resolve are indeed the bottleneck of QDA and TC.

As expected, for all the threshold values and all the answer semantics, QDA is both faster (than TC) and issues fewer resolves (than TC). This is since the query-awareness gives QDA the ability to exploit the in-preserving predicate property to add some records to the result-set without the need to resolve their corresponding edges.

In Figure 7, QDA takes only 0.004 seconds when $t = 1$ and all records satisfy the query threshold, whereas TC takes 0.52 seconds. This large saving happens because for $t = 1$ QDA will label all nodes as `in` and will not issue any calls to resolve. However, for difficult thresholds, e.g., $t = 128$, most nodes are labeled `maybe` and there are many potential cliques that can be added up to 128, which need to be resolved. Thus, QDA resolves 1770 edges and spends 0.2 seconds to answer the query, while TC takes 0.52 seconds. Note that the number of resolves issued (and thus the time spent) is related to the the number of potential cliques that may satisfy the query. That is, whenever the number of potential cliques that may satisfy the query decreases, the number of calls to resolve (and obviously the time) will decrease and vice versa.

Figure 9 presents the end-to-end running time and the number of resolves called by QDA using (more strict) distinct and exact answer semantics. QDA computes *distinct semantics* by first computing the initial result set RS using QDA with representative semantics, then it de-duplicates RS. The larger the cardinality of RS, the higher the extra cost is. For instance, when RS is large (e.g., $t \geq 1$) the number of resolve calls is also large. QDA with the *exact semantics* goes one step further and resolves all edges between the records in RS and the remaining records and thus it is more expensive than QDA for distinct semantics.

Finally, note that techniques like [6], discussed in Section 2, are not meant and not designed to optimize for queries with numeric attributes. Unlike QDA, they cannot avoid calling resolves and hence will not be able to outperform TC (and thus QDA).

Experiment 2 (QDA Speed Up). Figure 10 plots the speed up of QDA (using representative semantics) over TC for 5 different query types using 4 different threshold values. The QDA's speed up over TC is calculated as the end-to-end running time of TC divided by that of QDA.

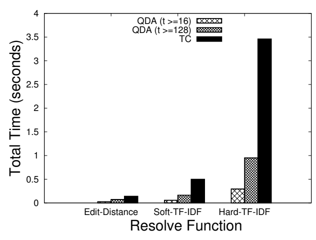

Figure 11: $\Re()$ **cost**

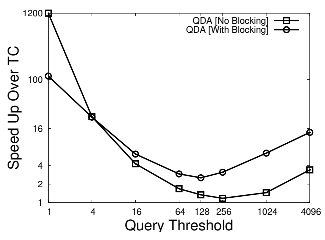

Figure 12: Blocking [Time]

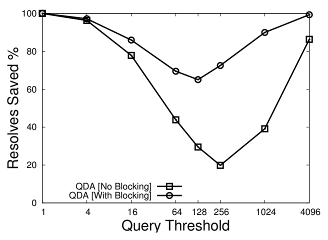

Figure 13: Blocking [%$\Re()$**]**

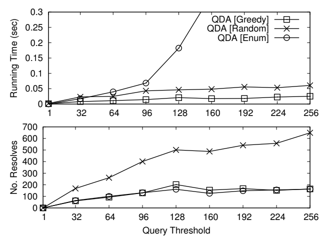

Figure 14: Edge picking

Note that QDA (for all queries and thresholds) is always faster than TC since QDA exploits the query to avoid calling some resolves while TC resolves each edge it picks. Depending on the query type and the threshold value, QDA can be from 1.2 to 100 times faster than TC.

As discussed in Experiment 1, the cost of calling resolve is the dominant factor of the overall execution time. Thus, the end-to-end execution time of QDA depends on the number of potential cliques which may satisfy the query since these potential cliques must be resolved.

For instance, QDA for LTE ($\leq$) takes slightly more time compared to QDA for GTE (yet, still 1.5 to 6 times faster than TC) because QDA using LTE can not exploit the in-preserving property to add records that satisfy the query threshold to the answer set. However, it now exploits the out-preserving property to remove records from the result-set. In this case, some of the edges connected to these discarded nodes need to be resolved because they might remove records from the result-set if they are declared duplicates.

The EQ ($=$) predicate is very selective and the number of *relevant* cliques that may satisfy the query is much smaller than that for GTE and LTE cases. In the EQ case, a *relevant* clique is either a clique that adds up to the value of the query threshold t or a clique that contains a sub-clique which adds up to t and at least one more node with a value greater than 0. However, this is the well-known NP-hard Subset Sum Problem and thus we implemented a $\frac{3}{4}$ linear approximation algorithm to find an approximate subset sum ($\simeq t$). This approximation lead to some unnecessary calls to resolve, and thus more time, causing QDA for EQ to be slightly more expensive compared to QDA for GTE. QDA for EQ is 1.5 to 6 times faster than TC.

In Figure 10, range queries are tested using the predicate $p : t - 50 \leq \mathit{cited} \leq t$. Recall that range queries are neither in- nor out-preserving (see Table 5). QDA for range queries is 1.2 to 6 times faster than TC. It takes a bit more time compared to QDA using LTE since the number of potential cliques which may change the query answer in QDA for range queries is slightly higher because one potential clique may put a record (say r_i) in the answer and then another potential clique may remove r_i from it.

Finally, in Figure 10 the predicate utilized to test categorical queries is $p : \mathit{cited} \geq t \wedge \mathit{venue} =$ 'VLDB'. The number of potential cliques which satisfy the query in this case is much smaller when compared to all previous cases (viz., GTE, LTE, EQ, and Range) because p is very selective. On the other hand, QDA spends more time checking for such cliques since they involve a categorical attribute. QDA for categorical queries is 7 to 10 times faster than TC.

Experiment 3 (Resolve Cost). Figure 11 demonstrates the importance of minimizing the number of calls to resolve, especially when the resolve function is not cheap. This experiment uses a smaller dataset of 448 publications written by a prolific CS professor and tests 3 different resolve functions of various costs. Function one is the least expensive and uses a normalized edit-distance function to compare titles and authors. The second function is more expensive and calculates Soft-TF-IDF for the titles and Jaro-Winkler distance between the authors. The third one is the most expensive: it computes TF-IDF for the abstracts of the papers. Note that in general, modern resolve functions can be even more expensive, e.g. involving ontology matching, web queries, etc. Figure 11 demonstrates that the gap between QDA and TC increases when the cost of $\Re()$ increases. Thus, minimizing the number of resolves is very important specifically for non-cheap resolve functions.

Experiment 4 (Applying Blocking). Figure 12 and 13 study the effects of using/not-using blocking on both QDA and TC. Figure 12 plots the speed up of QDA over TC and Figure 13 shows the percentage of resolves saved by using QDA instead of TC. Note that when no blocking is applied, all publications of an author are put in *one* block.

As expected, QDA outperforms TC according in both comparison criteria, for all threshold values – even when no blocking is applied. However, QDA's performance with blocking is better than its performance without blocking. This is since blocking removes some edges from consideration, thus causing the number of potential cliques which satisfy t to decrease dramatically.

An interesting case is when $t = 1$ and thus all records satisfy the query threshold. In that case, QDA without blocking is 1200 times faster than TC without blocking; whereas QDA with blocking is 100 times faster than TC with blocking. This is because, for $t = 1$, QDA with blocking take comparable amount of time to QDA without blocking, whereas TC with blocking is much faster than TC without blocking.

Experiment 5 (Edge Picking Strategy). Figure 14 studies the effectiveness of our edge-picking strategy. It compares three different strategies in terms of their end-to-end execution time and the number of calls to resolve: (1) our greedy policy, which chooses edges with higher weights first, (2) a random policy, which selects edges randomly, (3) an enumeration policy that enumerates all minimal cliques and chooses the edge involved in the maximum number of such cliques. Since the third approach is computationally very expensive, we had to conduct this test on a smaller dataset of 177 papers, written by the same author, to make sure that the test terminates in a reasonable amount of time.

As expected and shown in Figure 14, the third strategy tends to be very competitive in terms of the number of resolves called, as it quickly reduces the edge-search space. However, it is by far the worst strategy in terms of the end-to-end execution time. This is because enumerating all minimal cliques is computationally very expensive. In other words, this policy finds good edges, but it spends way too much time to find them.

Thus, our proposed greedy policy surpasses all other techniques: it not only finds good edges that are able to quickly reduce the edge-search space, it also finds them very quickly.

7.3 Discussion

Here we summarize a few other interesting experiments; their full versions are not included due to the page limit.

Analysis of QDA$^+$($\geq$/ADD). QDA$^+$ is a QDA approach for TC$^+$, instead of TC. TC$^+$ is a version of TC which treats MustSeparate *softly*. That is, when $\Re(p_i, p_j)$ returns MustSeparate, p_i and p_j are not merged at the moment due to a lack of evidence supporting a merge, but clusters containing p_i and p_j may be merged later due to transitivity.

Let us consider the performance of QDA$^+$, by looking at few cases. For the triple ($cited \geq 64$, ADD, $cited$), the percentage of resolves saved by QDA$^+$ over TC$^+$ is 46.4% whereas QDA saves 69.46% of resolves over TC. For $p : t \geq 128$, the savings percentage for QDA$^+$ is 30.07% while it is 65.04% for QDA. This lower achieved gain is expected, since QDA can merge (or partition) the graph by exploiting the yes/no edges, whereas QDA$^+$ can only merge nodes.

Test of MAX semantics. This experiment tests the MAX combine function for GTE queries. Note that this case is trivial, as for all threshold values no node is labeled maybe. That is, a node either satisfies t and hence is labeled in; or does not satisfy t and thus is labeled out. There are *no* two nodes which can be combined to change the result-set. Thus, QDA answers such queries very efficiently, without making a single call to resolve.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have studied the Query-Driven Entity Resolution problem in which data is cleaned "on-the-fly" in the context of a query. We have developed a *query-driven* entity resolution framework which efficiently issues the minimal number of cleaning steps solely needed to accurately answer the given selection query. We formalized the problem of query-driven ER and showed empirically how certain cleaning steps can be avoided based on the nature of the query. This research opens several interesting directions for future investigation. While selection queries (as studied in this paper) are an important class of queries on their own, developing QDA techniques for other types of queries (e.g., joins) is an interesting direction for future work. Another direction is developing solutions for efficient maintenance of a database state for subsequent querying.

9. ACKNOWLEDGEMENTS

This work was supported in part by NSF grants CNS-1118114, CNS-1059436, CNS-1063596. It is part of NSF supported project *Sherlock* @ UCI [2]. Hotham Altwaijry was supported by KACST's Graduate Studies Scholarship.

10. REFERENCES

[1] http://ics.uci.edu/~haltwaij/QDA.pdf.

[2] http://sherlock.ics.uci.edu.

[3] http://cs.ucla.edu/~palsberg/h-number.html.

[4] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pp. 586–597, 2002.

[5] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *VLDB J.*, pp. 255–276, 2008.

[6] I. Bhattacharya and L. Getoor. Query-time entity resolution. *JAIR*, pp. 621–657, 2007.

[7] M. Bilenko, S. Basil, and M. Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *DMKD*, pp. 8–pp, 2005.

[8] S. Chen, D. V. Kalashnikov, and S. Mehrotra. Adaptive graph- ical approach to entity resolution. In *JCDL*, pp. 204–213, 2007.

[9] Z. Chen, D. V. Kalashnikov, and S. Mehrotra. Exploiting context analysis for combining multiple entity resolution systems. In *SIGMOD*, pp. 207–218, 2009.

[10] W. W. Cohen, P. Ravikumar, S. E. Fienberg, et al. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pp. 73–78, 2003.

[11] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, pp. 85–96, 2005.

[12] E. Elmacioglu, M.-Y. Kan, D. Lee, and Y. Zhang. Web based linkage. In *WIDM*, pp. 121–128, 2007.

[13] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. In *KDE*, pp. 1-16, 2007.

[14] W. Fan, X. Jia, J. Lo, and S. Ma. Reasoning about record matching rules. In *VLDB*, pp. 407-418, 2009.

[15] I. P. Fellegi and A. B. Sunter. A theory for record linkage. In *JASA*, pp. 1183-1210, 1969.

[16] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pp. 127–138, 1995.

[17] T. Herzog, F. Scheuren, and W. Winkler. Data quality and record linkage techniques. In *Springer Verlag*, 2007.

[18] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. In *VLDB End.*, pp. 429–438, 2010.

[19] M. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. In *JASA*, pp. 414–420, 1989.

[20] D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. In *TODS*, pp. 716–767, 2006.

[21] P. Kanani, A. McCallum, and C. Pal. Improving author coreference by resource-bounded information gathering from the web. In *IJCAI*, pp. 429–434, 2007.

[22] A. K. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *SIGKDD*, pp. 169–178, 2000.

[23] H. Newcombe, J. Kennedy, S. Axford, and A. James. Automatic linkage of vital records. In *Science*, 1959.

[24] R. Nuray-Turan, D. V. Kalashnikov, and S. Mehrotra. Exploiting web querying for web people search. *TODS*, pp. 7–pp, 2012.

[25] Y. Sismanis, L. Wang, A. Fuxman, P. J. Haas, and B. Reinwald. Resolution-aware query answering for business intelligence. In *ICDE*, pp. 976–987, 2009.

[26] W. Su, J. Wang, and F. H. Lochovsky. Record matching over query results from multiple web databases. In *KDE*, pp. 578–589, 2010.

[27] S. E. Whang and H. Garcia-Molina. Entity resolution with evolving rules. In *PVLDB*, pp. 1326–1337, 2010.

[28] M. Yakout, A. K. Elmagarmid, H. Elmelegy, M. Ouzzani, and A. Qi. Behavior based record linkage. In *VLDB*, pp. 439–448, 2010.

[29] L. Zhang, D. V. Kalashnikov, and S. Mehrotra. A unified framework for context assisted face clustering. In *ICMR*, pp. 9–16, 2013.