QDA: A Query-Driven Approach to Entity Resolution

Hotham Altwaijry, Dmitri V. Kalashnikov, and Sharad Mehrotra, *Member, IEEE,*

Abstract—This paper addresses the problem of query-aware data cleaning in the context of a user query. In particular, we develop a novel *Query-Driven Approach* (QDA) that systematically exploits the semantics of the predicates in SQL-like selection queries to reduce the data cleaning overhead. The objective of QDA is to issue the minimum number of cleaning steps that are necessary to answer a given SQL-like selection correctly. The comprehensive empirical evaluation of QDA demonstrates outstanding results – that is QDA is significantly better compared to traditional ER techniques, especially when the query is very selective.

Index Terms—Query-driven approach, QDA, query-aware, entity resolution, SQL selection queries.

1 Introduction

THIS paper addresses the problem of *query-aware* data cleaning, wherein the needs of the query dictates which parts of the data should be cleaned. Query-aware cleaning is emerging as a new paradigm for data cleaning to support today's increasing demand for (near) real-time analytical applications. Modern enterprises have access to potentially limitless data sources, e.g., web data repositories, social media posts, clickstream data, etc. Analysts usually wish to integrate one or more such data sources (possibly with their own data) to perform joint analysis and decision making. As a result of merging data from different sources, a given real-world object may often have multiple representations, resulting in data quality challenges. In this paper, we focus on the Entity Resolution (ER) challenge [16], [19], [29].

Traditionally, entity resolution is performed in the context of data warehousing as an offline preprocessing step prior to making data available to analysis – an approach that works well under standard settings. Such an offline strategy, however, is not viable in emerging applications that need to analyze only small portions of the entire dataset and produce answers in (near) real-time [8], [23].

A query-driven approach is motivated by several key perspectives. First, the need for (near) real-time analysis requires modern applications to execute up-to-the-minute analytical tasks, making it impossible for those applications to use time-consuming standard back-end cleaning technologies. Second, in the case of data analysis scenario (e.g., queries on *online data*), where a data analyst may discover and analyze data as part of a single integrated step, the system will know "what to clean" only at query time (while the analyst is waiting to analyze the data). Last, a scenario wherein a *small* organization possesses a very large dataset, but needs to analyze only small portions of it to answer some analytical queries quickly. In such a case, it would be counterproductive for that organization to spend their limited computational resources on cleaning all the data, especially given that most of it is going to be unnecessary.

Recent work on query-aware ER have been proposed in the literature [7], [31], [32]. While such solutions address query-aware ER, they are limited to mention-matching and/or numerical aggregation queries executed on top of dirty data. Data analysis, however, often requires a different type of queries requiring SQL-style selections. For instance, a user interested in *only* well-cited (e.g., with citation count above 45) papers written by "Alon Halevy". In contrast to our work, the previous approaches cannot exploit the semantics of such a selection predicate to reduce cleaning.

To address these new cleaning challenges we proposed a *Query-Driven Approach* (QDA) to data cleaning [2]. QDA is an entirely new complementary paradigm for improving the efficiency: it is different from blocking [19], [24], [28] and is typically much more effective in conjunction with blocking. Given a block B, and an arbitrary complex selection predicate P, QDA analyzes which entity pairs do not need to be resolved to identify all entities in B that satisfy P. It does so by modeling entities in B as a graph and resolving edges (potentially) belonging to cliques that may change the query answer. QDA computes answers that are equivalent to those obtained by first using a regular cleaning algorithm, and then querying on top of the cleaned data. However, in many cases QDA computes such answers much more efficiently. A key concept driving QDA is that of *vestigiality*. A cleaning step (i.e., call to resolve) is vestigial (i.e., unnecessary) if QDA can guarantee that it can still compute a correct final answer without knowing the outcome of this resolve.

This paper extends significantly our previous work [2] in several directions. First, while previously we introduced the concept of vestigiality for a large class of SQL selection queries and developed techniques to identify vestigial cleaning steps; in this paper, we formally develop the concept of vestigiality. In particular, we (i) differentiate vestigiality from minimality and (ii) provide a theoretical study of the conditions under which vestigiality can be tested using cliques. Second, we significantly extend the discussion on the concept of triples (a triple $(p, \oplus, a_\ell)$ contains three components: a predicate p, a combine function $\oplus$, an attribute

- *Hotham Altwaijry is at King Abdulaziz City for Science & Technology.*
- *Dmitri V. Kalashnikov is at At&t Labs Research.*
- *Sharad Mehrotra is at University of California, Irvine (UCI).*

a_ℓ in which $\oplus$ is defined over) by providing formal lemmas and proofs for the general case where we combine two (or more) triples. Third, we demonstrate that QDA is generic and can work with different types of clustering algorithms. Specifically, we explore how the *eagerness* of the chosen clustering algorithm affects the computational efficiency of QDA. In our initial work [2], we developed QDA to work with *eager* clustering techniques (viz., those techniques that make their merging decisions as soon as the resolve function returns a positive decision [6], [12]). In this paper, we generalize QDA to work with *lazy* clustering techniques (viz., those techniques that tend to delay their merging decisions until a final clustering step [5]). Note that such a generalization requires a significant different QDA approach compared to the one we previously proposed [2]. Fourth, we develop new ideas that optimize the processing of equality and range queries. Finally, we present a more comprehensive experimental evaluation by providing experiments for the new lazy approach and by using another real-world world dataset (from a different domain) to test our solutions.

2 Related Work

Entity resolution is a well-recognized data quality problem and has received considerable attention in the literature. A survey by Elmagarmid et al. [14] presents a thorough overview of the existing work in ER. We classify ER techniques into:

Traditional ER. A typical ER cycle consists of several phases of data transformations that include: *blocking*, *similarity computation*, *clustering*, and *merging* [21], which can be intermixed. The first phase is *blocking* which is a divide and conquer approach used for improving ER efficiency [19]. Often blocking partitions records into buckets [24] or canopies [28]. 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 [16], [19], [29]. Recently new approaches exploit new information sources such as analyzing context [4], [10], [35], exploiting relationships between entities [25], domain/integrity constraints [15], behaviors of entities [34], and external knowledge bases such as ontologies and web search engines [13], [26], [30]. The next ER phase is *clustering* where matching records are grouped together into clusters [5], [6], [12], [18]. Finally, the *merging phase* combines elements of each cluster into a single record.

Query-aware ER. Recent work on query-aware ER have been proposed in the literature [3], [7], [31], [32] of which methods of Altwaijry et al. [3] and Wang et al. [32] are the most related to our work. The QuERy approach of Altwaijry et al. [3] aims to efficiently and accurately answer *join* queries issued on top of *multiple* dirty relations. It works as follows: given sets of blocks $\mathcal{B}^R, \mathcal{B}^S, \ldots$ and a complex join predicate P, QuERy analyzes which block pairs do join and hence, need to be cleaned. It only dictates when a block should be cleaned and is agnostic to how the block is actually cleaned. As a result, QuERy operates at macro (block) level: should a block be cleaned or not. In contrast, QDA aims to reduce the number of cleaning steps that are necessary to exactly answer *selection* queries spanning a

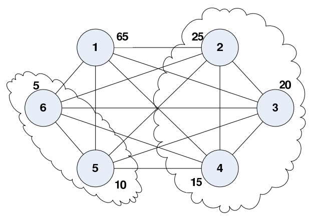

Fig. 1. Graph G

single dirty relation. In particular, it proposes algorithms for cleaning entities within a block. It operates at micro (entity pair) level: should an entity pair inside a block be resolved or not. Note that, in theory, QuERy could leverage QDA to be even more efficient by exploiting vestigiality analysis from the latter at the block level to reduce the number of entity pairs that are resolved within a block.

The SampleClean approach of Wang et al. [32] is designed to answer aggregate numerical queries over large datasets that cannot be fully cleaned. It focuses on cleaning only a sample of data and utilizing that sample to provide *approximate* answers to *aggregate* queries. It does not prune cleaning steps due to query predicates. Yet, QDA deals with *exact* answers to *selection* queries based on cleaning only the necessary parts of data needed to answer the query.

3 Notation and Problem Definition

In this section, we first present common ER notation, and then discuss new QDA-specific notation and formally define our problem.

3.1 Standard Notation

Relation and Clustering. Let $R = \{r_1, r_2, \ldots, r_{|R|}\}$ 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 $\langle a_1, a_2, \ldots, a_n \rangle$, where n is the arity of R. Thus, the k^{th} record in R is defined as $r_k = \langle \nu_{k1}, \nu_{k2}, \ldots, \nu_{kn} \rangle$, 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 $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ such that all records in each cluster refer to the same real-world entity and no two distinct clusters correspond to the same entity.

Graphical View of the Problem. The clustering problem can be represented graphically, as shown by Kalashnikov et al. [25] and Chen et al. [9], 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 4.3, QDA will create a much simplified version of this graph. For example, in Figure 1, we show graph G that represents a relation with 6 publication records. Such a relation is dirty since papers $\{p_2, p_3, p_4\}$ and $\{p_5, p_6\}$

are duplicates. Note that the numbers outside the nodes represent the *cited* count for each paper.

Resolve Function. A pairwise *resolve* function $\mathfrak{R}(r_i, r_j)$ 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) $\mathfrak{R}(r_i, r_j)$ = `MustMerge`, if resolve is highly confident r_i and r_j are the same and hence, must be merged,
2) $\mathfrak{R}(r_i, r_j)$ = `MustSeparate`, if resolve is highly confident r_i and r_j are different and hence, must be separated,
3) $\mathfrak{R}(r_i, r_j)$ = `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.

Naturally, the resolve function may output decisions that are *incorrect* and that could lead to mistakes in the entity resolution process.

Merge and Combine Functions. If $\mathfrak{R}(r_i, r_j)$ returns `MustMerge`, then the two records are declared to be duplicates and a *merge* function 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.

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

In general, the `ADD` semantics could be used when the objects are obtained from the same data source, yet their entities are split in parts. In this paper, `ADD` semantics are used as the default semantics to illustrate various examples, unless stated otherwise. The `MAX` and `MIN` semantics could be used when objects are retrieved from different data sources

If a_ℓ is a categorical attribute, then we consider:

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

The `UNION` semantics are utilized when the system needs to retain all possible values of some attribute. In contrast, the `EXEMPLAR` semantics are used when one value holds richer information than the other one.

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.

3.2 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 4.

Returned Answer Equivalence. Before we formally define various answer semantics, which allow `QDA` to provide a trade-off between the strictness of the chosen query answer semantics and the efficiency of query processing, we need to introduce several auxiliary concepts. Recall that a cluster C can be viewed as a set of records $C = \{r_1, r_2, \ldots, r_{|C|}\}$.

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 $\mathcal{A}$ 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 $\mathcal{C}_{\mathcal{A},Q}$ denote the set of clusters from $\mathcal{C}_{\mathcal{A}}$ that satisfy query Q. Let $\mathcal{C}_{\texttt{QDA},Q}$ 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 [33]. 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 $\mathcal{C}_{\texttt{QDA},Q}$ is exactly, distinctly, or representationally equivalent to an answer of $\mathcal{A}$ to query Q:

Definition 2. Answer $\mathcal{C}_{\texttt{QDA},Q}$ 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: (i) for each cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ there exists *exactly one* cluster $C_j \in \mathcal{C}_{\texttt{QDA},Q}$ such that $C_i \equiv C_j$, and (ii) for each cluster $C_j \in \mathcal{C}_{\texttt{QDA},Q}$ 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 $\mathcal{C}_{\texttt{QDA},Q}$ and $\mathcal{C}_{\mathcal{A},Q}$ and the content of clusters in $\mathcal{C}_{\texttt{QDA},Q}$ is *identical* to those in $\mathcal{C}_{\mathcal{A},Q}$.

We define the less restrictive *distinct* semantics as:

Definition 3. Answer $\mathcal{C}_{\texttt{QDA},Q}$ 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: (i) for each cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ there exists *exactly one* cluster $C_j \in \mathcal{C}_{\texttt{QDA},Q}$ such that $C_i \supseteq C_j$, and (ii) for each cluster $C_j \in \mathcal{C}_{\texttt{QDA},Q}$ 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 $\mathcal{C}_{\texttt{QDA},Q}$ and $\mathcal{C}_{\mathcal{A},Q}$, but now clusters in $\mathcal{C}_{\texttt{QDA},Q}$ are allowed to be *subsets* of clusters from $\mathcal{C}_{\mathcal{A},Q}$.

We define the least restrictive *representative* semantics as:

Definition 4. Answer $\mathcal{C}_{\texttt{QDA},Q}$ 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: (i) for each cluster $C_i \in \mathcal{C}_{\mathcal{A},Q}$ there exists *at least one* cluster $C_j \in \mathcal{C}_{\texttt{QDA},Q}$ such that $C_i \supseteq C_j$, and (ii) for each cluster $C_j \in \mathcal{C}_{\texttt{QDA},Q}$ 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 $\mathcal{C}_{\mathtt{QDA},Q}$ and one-to-one mapping from $\mathcal{C}_{\mathtt{QDA},Q}$ to $\mathcal{C}_{\mathcal{A},Q}$.

Problem Definition. Let $\mathcal{A}$ be the original entity resolution algorithm 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 $\mathfrak{R}()$

Subject to:

1. $\forall C \in \mathcal{C}_{\mathtt{QDA},Q}$, C satisfies Q; // *Query satisfaction*
2. $\mathcal{C}_{\mathtt{QDA},Q} \equiv \mathcal{C}_{\mathcal{A},Q}$; // *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 5.

4 Vestigiality Formulation

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 4.1. Then, we discuss how to deal with multi-predicate selection queries in Section 4.2. The construction of the labeled graph is explained in Section 4.3. We describe in Section 4.4, 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. Finally, in Section 4.5, we discuss the difference between vestigiality and minimality.

4.1 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. To achieve this goal, we classify any triple $(p, \oplus, a_\ell)$ into one of three 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, $(cited \geq 45, \mathtt{ADD}, cited)$ 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, $(cited \leq 45, \mathtt{ADD}, cited)$ is not in-preserving.

TABLE 1
Triple categorization for different types of attributes

$\oplus$, domain	$a_\ell \geq t$ or, $a_\ell > t$	$a_\ell \leq t$ or, $a_\ell < t$	$a_\ell = t$
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
EXEMPLAR, $a_\ell \in$ enum			in-preserving
UNION, $a_\ell \in$ enum			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., $(cited \leq 45, \mathtt{ADD}, cited)$ is out-preserving.

Table 1 shows a classification of different common triples for numerical and categorical attributes, respectively.

4.2 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 because 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 Figure 2. For instance, consider the following *range* query:

Query 1. SELECT * FROM R WHERE $cited \geq 45$ AND $cited \leq 65$

This range query consists of two basic predicates $p_1 : cited \geq 45$ and $p_2 : cited \leq 65$. Hence, it consists of two basic triples: an in-preserving triple $\tau_1 = (cited \geq 45, \mathtt{ADD}, cited)$ and an out-preserving triple $\tau_2 = (cited \leq 65, \mathtt{ADD}, cited)$. From Table 2, 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 1. 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.

We can prove a lemma for each combination (in Table 2) and following are the most important cases which extend the concepts of triples to more general predicates. Note that the proofs of these lemmas are provided in Appendix A.

Lemma 1. If triple $\tau_i = (p_i, \oplus_i, a_i)$ is in-preserving and triple $\tau_j = (p_j, \oplus_j, a_j)$ is in-preserving then, triple τ that results from the combination $\tau_i \wedge \tau_j$ is also in-preserving.

Lemma 2. If triple $\tau_i = (p_i, \oplus_i, a_i)$ is out-preserving and triple $\tau_j = (p_j, \oplus_j, a_j)$ is out-preserving, then triple τ that results from the combination $\tau_i \wedge \tau_j$ is also out-preserving.

Lemma 3. If triple $\tau_i = (p_i, \oplus_i, a_i)$ is in-preserving and triple $\tau_j = (p_j, \oplus_j, a_j)$ is out-preserving, then triple τ that results from the combination $\tau_i \wedge \tau_j$ is neither in- nor out-preserving.

Lemma 4. If triple $\tau_i = (p_i, \oplus_i, a_i)$ is in-preserving and triple $\tau_j = (p_j, \oplus_j, a_j)$ is neither in- nor out-preserving, then triple

TABLE 2
Triples generalization

τ_i	τ_j	$\tau_i \wedge \tau_j$	$\tau_i \vee \tau_j$	$\neg\tau_i$
in-preserving	in-preserving	in-preserving	in-preserving	out-preserving
in-preserving	out-preserving	neither	neither	out-preserving
out-preserving	in-preserving	neither	neither	in-preserving
out-preserving	out-preserving	out-preserving	out-preserving	in-preserving
in-preserving	neither	neither	neither	out-preserving
neither	in-preserving	neither	neither	neither
out-preserving	neither	neither	neither	in-preserving
neither	out-preserving	neither	neither	neither
neither	neither	neither	neither	neither

Algorithm 1 Create-Graph

```
1: Input: a set of records $R$, and a query $Q$
2: Output: a labeled graph $G$
3: $A_{cur} \leftarrow V_{out} \leftarrow V_{maybe} \leftarrow \{\}$
4: $\mathcal{B} \leftarrow$ CREATE-BLOCKS($R$)
5: for each $B \in \mathcal{B}$ do
6:   for each $r_k \in B$ do
7:     $v_k \leftarrow$ CREATE-NODE($r_k$)
8:     if IS-IN-PRESERVING($p, \oplus, a_\ell$) & SATISFY-Q($v_k, Q$) then
9:       $A_{cur} \leftarrow A_{cur} \cup \{v_k\}$        ▷ node $v_k$ is labeled in
10:    else if IS-OUT-PRESERVING($p, \oplus, a_\ell$) & not SATISFY-Q($v_k, Q$) then
11:      $V_{out} \leftarrow V_{out} \cup \{v_k\}$        ▷ node $v_k$ is labeled out
12:    else $V_{maybe} \leftarrow V_{maybe} \cup \{v_k\}$        ▷ node $v_k$ is labeled maybe
13:  $V \leftarrow \{A_{cur}, V_{out}, V_{maybe}\}$
14:  $E \leftarrow$ CREATE-EDGES($V_{maybe}, V_{maybe}$)
15:  $E \leftarrow E \cup$ CREATE-EDGES($V_{out}, V_{maybe}$)
16: return $G(V, E)$
```

τ that results from the combination $\tau_i \wedge \tau_j$ is neither in- nor out-preserving.

Lemma 5. If triple $\tau_i = (p_i, \oplus_i, a_i)$ is in-preserving and triple $\tau_j = (p_j, \oplus_j, a_j)$ is in-preserving then, triple τ that results from the combination $\tau_i \vee \tau_j$ is also in-preserving.

Lemma 6. If triple $\tau_i = (p_i, \oplus_i, a_i)$ is in-preserving, then triple τ that results from the negation $\neg\tau_i$ is out-preserving.

4.3 Creating and Labeling the Graph

To formally define vestigiality testing, we need to explain how QDA builds and labels the graph, see Algorithm 1. The main goal of this algorithm 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, as most traditional ER approaches would; by partitioning records into (possibly overlapping) smaller blocks. Then, it iterates over each record $r_k \in B$ to create the corresponding node v_k. It sets label $\ell[v_k]$ of v_k 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 (i) only among nodes in V_{maybe} and (ii) for each v_i, v_j pair where $v_i \in V_{out}$ and $v_j \in V_{maybe}$. 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 its label as follows:

1) $\ell[e_{ij}]$ = yes, when $\mathfrak{R}(r_i, r_j)$ has already been called and returned MustMerge,
2) $\ell[e_{ij}]$ = no, when $\mathfrak{R}(r_i, r_j)$ has already been called and returned MustSeparate,
3) $\ell[e_{ij}]$ = maybe, when $\mathfrak{R}(r_i, r_j)$ 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 graph G 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 $\mathfrak{R}(p_2, p_3)$ and $\mathfrak{R}(p_4, p_5)$, the corresponding edges will be assigned labels $\ell[e_{23}]$ = yes and $\ell[e_{45}]$ = no.

4.4 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 7. 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 7 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.

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. Let us prove that if an edge e_{ij} is not vestigial, then it belongs to a relevant clique. By contradiction, if an edge does not belong to any relevant clique, then it cannot participate in changing $\mathcal{C}_{cur}$ by adding or removing one (or more) cluster from it, thus such an edge is `vestigial`. □

The next example describes the relevant cliques concept.

Example 2. Consider G shown in Figure 1 after resolving e_{23} only and an in-preserving triple ($cited \geq 45$, ADD, $cited$). $A_{cur} = \{p_1, p_2 \oplus p_3\}$ 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 $15+10+5 = 30 \ngeq 45$. Thus, merging nodes in S cannot change A_{cur}. 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. Let us first prove that if edge e_{ij} belongs to a minimal clique S_{min}, then it is not vestigial. Note that it is possible that all of the `unresolved` edges, except for those in S_{min}, are `no` edges. If that is the case, and if at least one of the edges in S_{min} is a `no` edge in the ground truth, then the nodes in S_{min} do not co-refer and $\mathcal{C}_{cur}$ is the correct final answer. Otherwise, if all of the edges in S_{min} are `yes` edges in the ground truth, then the nodes in S_{min} do co-refer and the correct final answer will be different from current answerA_{cur}, in which a new cluster C_i will be added to the set of clusters $\mathcal{C}_{cur}$ since triple $(p, \oplus, a_\ell)$ is in-preserving. Therefore, the algorithm cannot permanently ignore edge e_{ij} by treating it as a `no` edge, since it might influence the correctness of the answer which the algorithm produces. Hence, this edge is not vestigial.

Algorithm 2 Is-Vestigial

```
1: Input: an edge $e_{ij}$, a graph $G$, and a query $Q$
2: Output: true if $e_{ij}$ is vestigial, false otherwise
3: if IS-IN-PRESERVING($p, \oplus, a_\ell$) then
4:     return not IS-IN-A-MINIMAL-CLIQUE($e_{ij}, G, Q$)
5: else return not IS-IN-A-RELEVANT-CLIQUE($e_{ij}, G, Q$)
```

Let us now prove that if an edge e_{ij} is not vestigial, then it belongs to a minimal clique. By contradiction, if an edge does not belong to any relevant clique, then it cannot participate in forming a new cluster that would change $\mathcal{C}_{cur}$ and thus it is `vestigial`. If, however, an edge does belong to a relevant clique S, but not a minimal clique, then two cases are possible. If this is a `no` edge in the ground truth, then the algorithm will compute the correct answer by declaring this edge as `vestigial`, because all vestigial edges are treated as `no` edges in the final answer computations. If this is a `yes` edge, then two more cases are possible. Let S_{min} be a minimal clique that is part of S. Then, if the nodes in S_{min} do not co-refer, then it is safe to declare e_{ij} to be `vestigial` with respect to S_{min}, since S_{min} does not form a new cluster that must be accounted for in forming the final answer. If, however, the nodes in S_{min} do co-refer, then let C_i be the new cluster they represent. But the algorithm does not need to know the value of $\mathfrak{R}(e_{ij})$ to determine how to represent C_i in its final answer (it can and will determine that, instead, when resolving edges from S_{min}), and hence e_{ij} can be declared `vestigial`. □

The next example explains the minimal cliques concept.

Example 3. Consider G shown in Figure 1 after resolving e_{45} only. Note that the triple $(cited \geq 45, \text{ADD}, cited)$ 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 Algorithm 2. However, finding such cliques is NP-hard as shown in Theorem 3.

Theorem 3. Testing for vestigiality using Algorithm 2 is NP-hard. This can be shown through a straightforward reduction from the well-known k-clique problem, and hence is computationally infeasible.

Thus, implementing Algorithm 2 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.

4.5 Vestigiality Versus Minimality

Before presenting the difference between the concept of vestigiality and the concept of minimality, let us first define the concept of minimality.

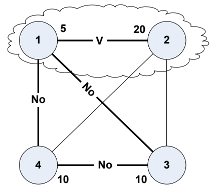

Fig. 2. Before resolving e_{12}

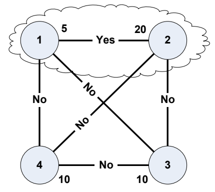

Fig. 3. After resolving e_{12}

Definition 11. An edge resolution sequence is said to *minimal*, if it is the shortest sequence that resolves a graph G to answer a query Q correctly.

In other words, the minimal edge resolution sequence is the most efficient way that resolves a graph to answer a given query. Even though, intuitively, avoiding resolving vestigial edges should lead to improved efficiency, it is possible that the minimal edge resolve sequence includes a vestigial edge, as stated by the following lemma:

Lemma 8. Given a labeled graph G, only resolving *non-vestigial* edges in G does not guarantee to lead to the minimal edge resolve sequence.

Proof. By example, consider the graph in Figure 2 that corresponds to three clusters $C_1 = \{v_1, v_2\}$, $C_2 = \{v_3\}$, and $C_3 = \{v_4\}$ and an in-preserving triple e.g., (*val* ≥ 30, ADD, *val*). Edge e_{12} is vestigial, since even if nodes v_1 and v_2 are the same, they can only form a new node with a citation count $5 + 20 = 25 \not\geq 30$. Whereas, the other two edges (viz., e_{23} and e_{24}) are not vestigial, since both can form nodes (i.e., $v_2 \oplus v_3$ or $v_2 \oplus v_4$) with citation count equal to 30. Querying for the vestigial edge e_{12}, however, would result in the graph shown in Figure 3, where edges e_{23} and e_{24} are no edges since we know that v_1 is different than v_3 and v_4. Hence, by issuing resolve once, the algorithm can compute the final answer, which is the empty set in this case. It is easy to see that if the algorithm starts by resolving any other edge instead of e_{12}, then one more resolve will be required. □

5 Query-Driven Solutions

In this section, we describe our query-driven solutions. We first present eager-QDA, which works with *eager* clustering techniques (viz., those techniques that make their merging decisions as soon as the resolve function returns a positive decision [6], [12]). Afterwards, we show lazy-QDA, which works with *lazy* clustering techniques (viz., those techniques that tend to delay their merging decisions until a final clustering step [5]). Finally, we study different optimizations to process equality and range style queries.

5.1 QDA Using Eager Clustering Techniques

The main task of QDA is to compute an answer to query Q very efficiently. The answer should be equivalent to first applying a standard ER algorithm on the whole dataset and then querying the resulting cleaned data with query Q.

In this section, we develop QDA to work with *eager* clustering techniques. Recall that a traditional eager ER algorithm (abbreviated eager-ER), which uses Transitive Closure Clustering [18] (as an example) to group matching entities together into clusters, 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. Our eager-QDA approach is very similar to eager-ER with two noticeable differences. First, eager-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, eager-QDA first tries to quickly determine if it can avoid making this call by checking if the chosen pair is vestigial.

Conceptually, eager-QDA 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 4.3).
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 eager-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 eager-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, eager-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 (i) v_i (or v_j) was removed from V_{maybe}, or (ii) 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 5.1.1).
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 5.1.2).

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

Algorithm 3 Vestigiality-Testing

```
1: Input: an edge $e_{ij}$, a graph $G$, and a query $Q$
2: Output: a labeled edge $e_{ij}$
3: if IS-IN-PRESERVING($p, \oplus, a_\ell$) & MIGHT-CHANGE-ANS($\varnothing, v_i \oplus v_j, Q$) then
4:     $res \leftarrow \Re(v_i, v_j)$
5:     if $res$ = MustMerge then
6:         $A_{cur} \leftarrow A_{cur} \cup \{v_i \oplus v_j\}$
7:         $V_{maybe} \leftarrow V_{maybe} - \{v_i, v_j\}$
8:     else if $res$ = MustSeparate then
9:         $E \leftarrow E - \{e_{ij}\}$
10:    else $\ell[e_{ij}]$ = maybe
11: else if CHECK-POTENTIAL-CLIQUE($e_{ij}, G, Q$) then
12:    $res \leftarrow \Re(v_i, v_j)$
13:    if $res$ = MustMerge then
14:        $v_i \leftarrow v_i \oplus v_j$
15:        $\mathcal{N}[v_i] = \mathcal{N}[v_i] \cap \mathcal{N}[v_j]$
16:        $V_{maybe} \leftarrow V_{maybe} - \{v_j\}$
17:    else if $res$ = MustSeparate then
18:        $E \leftarrow E - \{e_{ij}\}$
19:    else $\ell[e_{ij}]$ = maybe
20: else $E \leftarrow E - \{e_{ij}\}$                ▷ edge $e_{ij}$ is vestigial
```

simple strategy such as resolving all $O(n^2)$ edges in random order might be more efficient.

The next sections explain all of these steps in detail.

5.1.1 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 4, we know that testing for the exact vestigiality via clique-checking is an NP-hard problem. Hence, eager-`QDA` employs a highly efficient approximate solution instead of the exact check. Namely, eager-`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.

Algorithm 3, can conceptually be viewed as consisting of the following steps:

1) *Edge Minclique Check Optimization.* This is another optimization which the algorithm employs. 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 algorithm calls Algorithm 4 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. If the call returns `false`, then the function marks e_{ij} as `vestigial`.

The key intuition behind Algorithm 4 is to *quickly* check if an edge e_{ij} can *potentially* be involved in a relevant/minimal clique at all. 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.

Algorithm 4 Check-Potential-Clique

```
1: Input: an edge $e_{ij}$, a graph $G$, and a query $Q$
2: Output: true if $e_{ij}$ is in a potential clique, false otherwise
3: if MIGHT-CHANGE-ANS($\varnothing, v_i \oplus v_j, Q$) then
4:     return true
5: $V_{intersect} \leftarrow \mathcal{N}[v_i] \cap \mathcal{N}[v_j]$
6: for each $v_k \in V_{intersect}$ do
7:     $v_{old} \leftarrow v_{new}$
8:     $v_{new} \leftarrow v_{old} \oplus v_k$
9:     if MIGHT-CHANGE-ANS($v_{old}, v_{new}, Q$) then
10:        return true
11: return false
```

The idea of this algorithm 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.

Algorithm 4 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 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`. Otherwise, it returns `false`.

5.1.2 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 Algorithm 5 which starts by adding nodes from V_{maybe} which satisfy Q to A_{cur}. 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 eager-ER.

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 eager-ER. That is, duplicates are removed by resolving all pairs of nodes in A_{cur}. 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 $A_{cur} \cup V_{maybe}$. 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* answers, edges formerly labeled `vestigial` are considered `unresolved`.

Algorithm 5 Compute-Answer

```
1:  Input: a current answer $A_{cur}$, a set of maybe nodes $V_{maybe}$, a query $Q$, and
    an answer semantic $S$
2:  Output: a clean final answer $A_{cur}$
3:  for each $v_i \in V_{maybe}$ do
4:    if SATISFY-Q($v_i, Q$) then
5:      $A_{cur} \leftarrow A_{cur} \cup v_i$
6:  if $S$ = Distinct then
7:    for each $v_i \in A_{cur}$ do
8:      for each $v_j \neq v_i \in A_{cur}$ do
9:        if $\mathfrak{R}(v_i, v_j)$ = MustMerge then
10:         $v_i \leftarrow v_i \oplus v_j$
11:         $A_{cur} \leftarrow A_{cur} - \{v_j\}$
12: else if $S$ = Exact then
13:   for each $v_i \in A_{cur}$ do
14:     for each $v_j \neq v_i \in A_{cur} \cup V_{maybe}$ do
15:       if $\mathfrak{R}(v_i, v_j)$ = MustMerge then
16:         $v_i \leftarrow v_i \oplus v_j$
17:         if $v_j \in A_{cur}$ then
18:           $A_{cur} \leftarrow A_{cur} - \{v_j\}$
```

5.1.3 *Answer Correctness*

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

Lemma 9. If the resolve function is always accurate, then eager-QDA will compute answers that are: *representationally*, *distinctly*, or *exactly* equivalent to those in $\mathcal{C}_{gt}$.

Lemma 9 essentially states a theoretical result: eager-QDA 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.

5.2 QDA Using Lazy Clustering Techniques

In this section, we demonstrate that our query-driven solution is generic and can work with different types of clustering algorithms. In particular, herein, we develop QDA to work with *lazy* clustering techniques (i.e., those techniques that tend to delay their merging decisions until a final clustering step). Lazy-QDA uses (as an example) a variation of the *Cautious* Correlation Clustering algorithm to minimize disagreements by Bansal et al. (denoted by CC) to group matching entities into clusters. The overall clustering problem is represented as a fully connected graph, where each object becomes a node in the graph. Each edge e_{ij} is assigned a + (similar) or − (different) label, according to the resolve function $\mathfrak{R}(r_i, r_j)$. Unlike eager-QDA which computes equivalent answers to those obtained by eager-ER, lazy-QDA cannot guarantee equivalence to an original lazy ER algorithm (abbreviated lazy-ER). However, it finds answers that are very close to those of lazy-ER while being more efficient as will show in our experiments. Note that, in general, developing lazy-QDA is more challenging, since lazy-ER needs to know the results of all $O(n^2)$ resolves.

The main idea of this solution is that lazy-QDA can be implemented by leveraging eager-QDA (see Section 5.1). Specifically, observe that if resolve cannot return Uncertain and always returns accurate answers, then eager-QDA is guaranteed to compute an answer equivalent to $\mathcal{C}_{gt}$ (see Lemma 7). Let $\mathcal{C}_{cc}$ be the clustering computed by lazy-ER, which uses CC to cluster matched entities. In addition, let us modify the resolve function on edge e_{ij} to return the same MustMerge/MustSeparate value as this edge e_{ij} has in $\mathcal{C}_{cc}$. Then, according to Lemma 7, eager-QDA with this new resolve function will compute an answer that is equivalent to $\mathcal{C}_{lazy}$ instead of $\mathcal{C}_{gt}$, which is exactly what we need.

The challenge is that we do not know what a lazy-ER will return as final clusters without calling all $O(n^2)$ resolves, which defeats the purpose of QDA. To *imitate* the result of lazy-ER without calling all resolves, lazy-QDA first calls eager-QDA but then *checks* whether the outcome of resolves called by eager-QDA could be different according to lazy-ER. If they cannot be, then eager-QDA is supposed to produce equivalent answer to that of lazy-ER.

To perform the above check, lazy-QDA needs to know if there exists some assignment (of MustMerge/MustSeparate values) of yet-unresolved (by eager-QDA) edges such that CC applied on graph G with that assignment could change the values of the resolves consumed by QDA. To do so, lazy-QDA applies a "stress test" – by using two CC extremes. The first extreme (denoted by CC$^+$) clusters records by assuming all unresolved edges are yes edges and then apply CC clustering normally. In contrast, the second extreme (denoted by CC$^-$) marks all unresolved edges as no edges and then executes CC. Intuitively, CC$^+$ tries to group all records into as few clusters as possible with the maximum force – given the known constraints for already resolved edges. In contrast, CC$^-$ tries to do the opposite – to split all records into as many clusters as possible with the maximum force. If the resolves called by eager-QDA cannot "flip" (e.g., from yes to no or from no to yes) according to these two stress tests, then the edges are considered "stable". This test is a *very good* heuristic to check for stable edges.

More precisely, lazy-QDA trusts clusterings obtained by both CC$^+$ and CC$^-$ (denoted by $\mathcal{C}^+$ and $\mathcal{C}^-$ respectively) rather than eager-QDA clustering. Thus, it compares the labeling of each *resolved* edge in $\mathcal{C}^+$ and $\mathcal{C}^-$. If both clusterings agree on edge e_{ij} labeling (e.g., $\ell[e^+_{ij}]$ = yes and $\ell[e^-_{ij}]$ = yes), then e_{ij} is stable and regardless of $\mathfrak{R}(v_i, v_j)$, e_{ij} is labeled as yes. Yet, if clusterings disagree (e.g., $\ell[e^+_{ij}]$ = yes and $\ell[e^-_{ij}]$ = no), then e_{ij} is *unstable* and hence labeled maybe. At the end, if all edges are stable then lazy-QDA terminates, if not then it proceeds to the next iteration.

In the next iteration, lazy-QDA will update G based on the new modifications from CC$^+$ and CC$^-$. For example, it will merge two nodes if the edge between them was deemed to be yes by both $\mathcal{C}^+$ and $\mathcal{C}^-$. However, if an edge between two records flipped from yes (or no) to maybe, then lazy-QDA will neither merge nor separate the two records. It will represent the edge as a maybe edge. To avoid an excessive number of iterations, lazy-QDA will try to overcome the destabilization in the graph caused by maybe edges by resolving other edges close to them. Note that lazy-QDA will not resolve an edge which had been already resolved. After that, lazy-QDA proceeds iteratively from the beginning. It terminates when all edges are stable.

5.3 Optimizations of Equality and Range Queries

In this paper, we have covered very generic algorithms that can be applied to a broad class of cases that are based on categorizing triples into in- and out-preserving and neither. However, various optimizations of these algorithms are possible when considering each specific case separately. In this section, we study a different implementation for equality and range style queries.

Queries with Equality Predicates. 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 > 0. However, this is the NP-hard Sub Set Problem and hence, at first, we implemented a $\frac{3}{4}$ linear approximation algorithm [27] to find an approximate subset sum ($\simeq t$). This approximation lead to some unnecessary calls to resolve causing QDA, in the EQ case, to issue more resolves than both GTE and LTE. However, an optimization which postpones the resolution of some edges (instead of greedily resolving them) allowed QDA to issue a slightly smaller number of resolves when compared to the GTE case and a much smaller number of resolves compared to the LTE case.

The intuition behind this optimization is three-fold. First, the algorithm does not create an edge between any two nodes labeled as out. This optimization is similar to an optimization implemented when the triple is out-preserving. However, the process of labeling nodes differs in the EQ case; here, only nodes with values greater than t are labeled as out whereas when the triple is out-preserving, any node that does not satisfy the query's condition is labeled as out.

Second, when trying to find the set of mutual neighbors, there might exist some nodes such that their values are greater than t. Such nodes can be discarded since they cannot lead to a sum value, denoted by ν_{sum}, equal to t. This idea can be further optimized by discarding all nodes with values greater than the *dynamic move past point* (denoted by *MPP*) such that $MPP = t - \nu_i - \nu_j$, where ν_i and ν_j are the values of edge e_{ij} endpoints since they also cannot contribute to ν_{sum} equal to t.

Third, after determining the set of mutual neighbors, the algorithm starts to sum them up. If it reaches a point where ν_{sum} is greater than or equal to t, it invokes a resolve call on edge e_{ij}. Otherwise, if $\nu_{sum} < t$, then it removes e_{ij} (lazily) from graph G (causing other edges to be removed too) and adds it to the *postponed edges* set (the other edges removed from G are not added to this set). At the end, the algorithm checks whether the edges in the postponed edges set can remove nodes from the answer set. This removal can happen if the value of one of the edge's endpoints is equal to t.

Queries with Range Predicates. The idea behind this optimization 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 *erroneous* nodes that should not be there. To remove these erroneous 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 improvement in processing range queries.

6 Experimental Evaluation

In this section, we empirically evaluate the efficiency of our QDA approaches on real and synthetic data.

In Section 6.1, we study our QDA solutions for different query types. In particular, we first compare eager-QDA with eager-ER in terms of the end-to-end running time and the number of calls to resolve. Note that both approaches use Transitive Closure Clustering to cluster the matched entities. In addition, we compare lazy-QDA with lazy-ER in terms of the number of resolve invocations. Note that both approaches utilize a variation of the minimizing disagreements Correlation Clustering to cluster the matched entities. We also study the quality reached by eager-QDA, eager-ER, lazy-QDA, and lazy-ER. We further test how close the results of lazy-QDA get to those of lazy-ER.

In Section 6.2, we study eager-QDA using more complex selection queries with multiple predicates connected via logical connectives. In particular, we study eager-QDA using the three generic categories of triples.

6.1 Google Scholar Dataset Experiments

In this section, we evaluate the efficacy of QDA on a real bibliographic dataset collected from Google Scholar. It represents publications of the top 50 computer science researchers each having h-index of 60 or higher [1]. The dataset consists of $16,396$ records where 14.3% are duplicates.

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. Note that both eager-QDA and eager-ER use the same blocking procedure.

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 [11] 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.

Experiment 1 (Eager-QDA vs. eager-ER). Figure 4 uses a set of GTE ($\geq$) queries of the form (SELECT * FROM R WHERE $cited \geq t$) to show the effects of vestigially testing by comparing eager-QDA (using representative semantics) with eager-ER. For each threshold t, we run 50 queries – one query for each author. Then, we sum up the time taken (or number of resolves invoked) by all these 50 queries and divide it by 50 to report the average running time (number of resolves called) for each threshold.

The top part of Figure 4 plots the actual end-to-end execution time while the bottom part plots the number of calls to resolve of eager-QDA and eager-ER for different values of the GTE query threshold t. These are log-lin scale plots, where t's range is chosen to show the entire spectrum

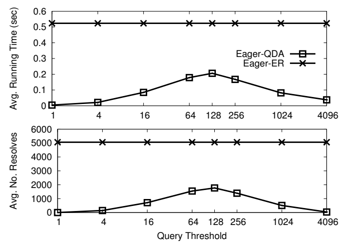

Fig. 4. Eager-QDA vs. eager-ER

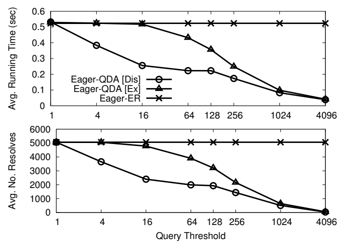

Fig. 5. Answer semantics

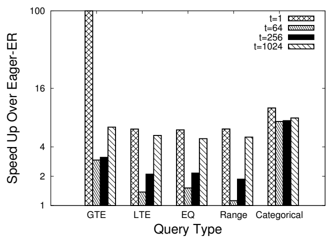

Fig. 6. Speed up of eager-QDA

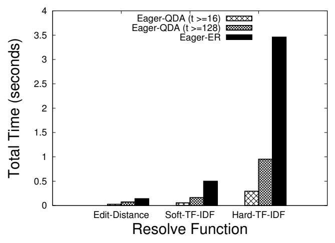

Fig. 7. $\mathfrak{R}()$ cost

of eager-QDA's behavior. Note that the curves in Figure 4 are similar, thus demonstrating the fact that the calls to resolve are indeed the bottleneck of eager-QDA and eager-ER.

As expected, for all the threshold values and all the answer semantics, eager-QDA is both faster and issues fewer resolves (than eager-ER). This is since the query-awareness gives eager-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 4, eager-QDA takes only 0.004 seconds when $t = 1$ and all records satisfy the query threshold, whereas eager-ER takes 0.52 seconds. This large saving happens because for $t = 1$ eager-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, eager-QDA resolves 1770 edges and spends 0.2 seconds to answer the query, while eager-ER 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.

Experiment 2 (Answer Semantics). Figure 5 presents the end-to-end running time and the number of resolves called by eager-QDA using (more strict) distinct and exact answer semantics. Eager-QDA computes *distinct semantics* by first computing the initial result set RS using eager-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. Eager-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 eager-QDA for distinct semantics.

Experiment 3 (Speed up of eager-QDA). Figure 6 plots the speed up of eager-QDA (using representative semantics) over eager-ER for 5 different query types using 4 different threshold values. The eager-QDA's speed up over eager-ER is calculated as the end-to-end running time of eager-ER divided by that of eager-QDA.

Note that eager-QDA (for all queries and thresholds) is always faster than eager-ER since eager-QDA exploits the query to avoid calling some resolves while eager-ER resolves all edges. Based on the query type and the threshold value, eager-QDA can be from 1.2 to 100 times faster than eager-ER.

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 eager-QDA depends on the number of potential cliques which may satisfy the query since these potential cliques must be resolved.

For instance, eager-QDA for LTE ($\leq$) takes slightly more time compared to eager-QDA for GTE (yet, still 1.5 to 6 times faster than eager-ER) as eager-QDA using LTE cannot exploit the in-preserving property to add records that satisfy the query threshold to the answer set. However, it exploits the out-preserving property to remove records from the answer. In this case, some of the edges connected to these discarded nodes need to be resolved because they might remove records from the answer 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 eager-QDA for GTE. Eager-QDA for EQ is 1.5 to 6 times faster than eager-ER.

In Figure 6, range queries are tested using the predicate $p : t - 50 \leq cited \leq t$. Recall that range queries are neither in- nor out-preserving (see Table 2). Eager-QDA for range queries is 1.2 to 6 times faster than eager-ER. It takes a bit more time compared to eager-QDA using LTE since the number of potential cliques which may change the query answer in eager-QDA for range queries is slightly higher because one potential clique may put record r_i in the answer and then another potential clique may remove r_i from it.

Finally, in Figure 6 the predicate utilized to test categorical queries is $p : cited \geq t \wedge venue =$ '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, eager-QDA spends more time checking for such cliques since they involve a categorical attribute. eager-QDA for categorical queries is 7 to 10 times faster than eager-ER.

Experiment 4 (Resolve Cost). Figure 7 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

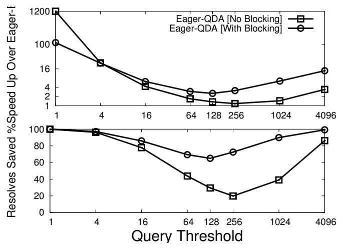

Fig. 8. Blocking

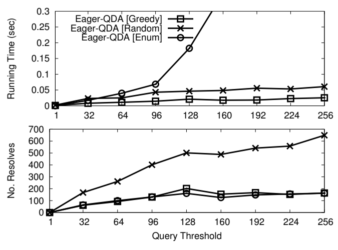

Fig. 9. Edge picking strategy

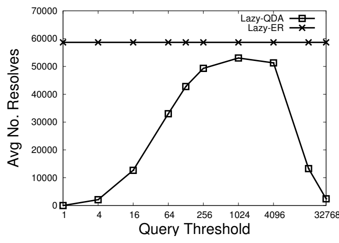

Fig. 10. Lazy-QDA vs. lazy-ER [#$\mathfrak{R}()$]

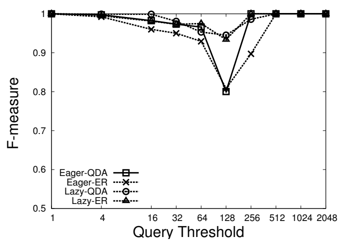

Fig. 11. Quality of algorithms

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 7 demonstrates that the gap between eager-QDA and eager-ER increases when the cost of $\mathfrak{R}()$ increases. Thus, minimizing the number of resolves is very important specifically for non-cheap resolve functions.

Experiment 5 (Applying Blocking). Figure 8 studies the effects of using/not-using blocking on both eager-QDA and eager-ER. The top part of Figure 8 plots the speed up of eager-QDA over eager-ER while the bottom part of that figure shows the percentage of resolves saved by using eager-QDA instead of eager-ER. Note that when no blocking is applied, all publications of an author are put in *one* block.

As expected, eager-QDA outperforms eager-ER according in both comparison criteria, for all threshold values – even when no blocking is applied. However, eager-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, eager-QDA without blocking is 1200 times faster than eager-ER without blocking; whereas eager-QDA with blocking is 100 times faster than eager-ER with blocking. This is because, for $t = 1$, eager-QDA with blocking take comparable amount of time to eager-QDA without blocking, whereas eager-ER with blocking is much faster than eager-ER without blocking.

Experiment 6 (Edge Picking Strategy). Figure 9 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: (i) our greedy policy, which chooses edges with higher weights first, (ii) a random policy, which selects edges randomly, (iii) 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 9, 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 greedy policy surpasses all other approaches: it not only finds good edges that are able to quickly reduce the edge-search space, it also finds them very quickly.

Experiment 7 (Lazy-QDA vs. lazy-ER). This experiment compares lazy-QDA with lazy-ER in terms of the number of resolve invocations. It uses the representative semantics to compute the final answer. Note that, in this experiment, we are being conservative by not using blocking (viz., *no blocking*, see Experiment 5) to show that lazy-QDA outperforms lazy-ER even in this difficult case.

Figure 10 shows that lazy-QDA issues less resolve compared to lazy-ER for all t values. Since lazy-QDA is cognizant of t, it marks some edges as vestigial without resolving them, whereas lazy-ER must resolve all $O(|E|^2)$ edges in the graph. Note that, for very large thresholds (e.g., $t = 16,384$), lazy-QDA resolves only a small number of edges in the first iteration. However, most of these few edges "flip" to maybe edges in the "CC^+ vs. CC^-" test. Hence, to avoid too many iterations, lazy-QDA stabilizes the graph by resolving edges that are close to the flipped edges.

Experiment 8 (Quality). Figure 11 plots the quality reached by eager-QDA, lazy-QDA, eager-ER, and lazy-ER when answering $cited \geq t$ query for various values of t. To compute quality, we use a standard quality metric called F-measure. Specifically, the precision of an ER algorithm $\mathcal{A}$ is computed as $Pr = \frac{|\mathcal{C}_{\mathcal{A},Q} \cap \mathcal{C}_{GT,Q}|}{|\mathcal{C}_{\mathcal{A},Q}|}$, where $\mathcal{C}_{\mathcal{A},Q}$ is the set of clusters returned by $\mathcal{A}$ that satisfy query Q, and $\mathcal{C}_{GT,Q}$ is the set of clusters in the ground truth which satisfy Q. The recall is computed as $Re = \frac{|\mathcal{C}_{\mathcal{A},Q} \cap \mathcal{C}_{GT,Q}|}{|\mathcal{C}_{GT,Q}|}$. The F-measure is computed as the harmonic mean of the precision and recall $F = \frac{2 \cdot Pr \cdot Re}{Pr + Re}$.

In order to measure the quality accurately, we use a synthetic dataset (where its ground truth is known) in this test. This dataset is generated using an enhanced version of the UIS data generator [20]. UIS has been applied on a clean dataset of 100 tuples to generate datasets of size 100 to 1000 tuples by injecting various types of errors in it. Note that we modified our resolve function $\mathfrak{R}()$ to make errors occasionally. That is, after resolving a pair of records and returning the decision, it reverses its decision intentionally to make a mistake. The percentage of reversed decisions (that is, errors) is 5%. We had to modify our resolve since otherwise all methods will be very accurate and all will

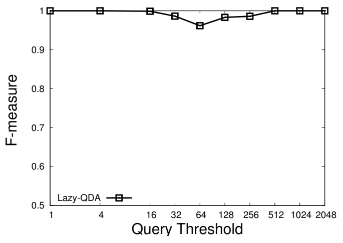

Fig. 12. Lazy-QDA deviation

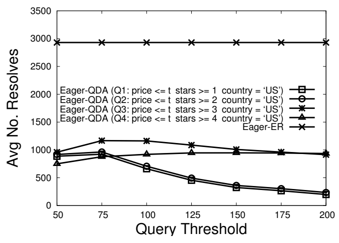

Fig. 13. In-preserving

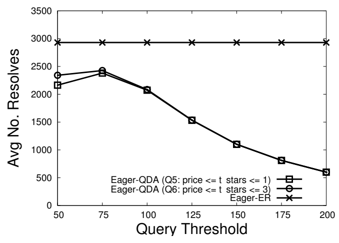

Fig. 14. Out-preserving

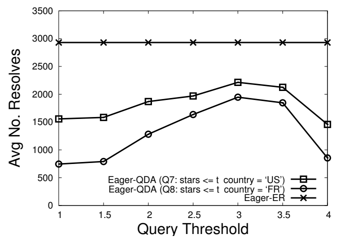

Fig. 15. Neither

reach nearly perfect quality. The majority of the introduced errors will *merge* distinct nodes.

From Figure 11 we observe that all algorithms reach very high quality for small (1 to 16) and large (≥ 512) thresholds. We also observe that the quality of eager-QDA and eager-ER drops significantly when $t = 128$ because of their eagerness-to-merge behavior. Recall that, most introduced errors *merge* distinct nodes and hence, construct an incorrect cluster that satisfies the query (i.e., causing a drop in Pr).

Note that in this plot the results of CC-based strategies (i.e., lazy-QDA and lazy-ER) turned out to be better than those of Transitive Closure-based (i.e., eager-QDA and eager-ER). However, this is not always the case in general and the reverse might be true for different datasets, e.g., see [18].

Experiment 9 (Lazy-QDA Deviation). In this test, we study how close the results of lazy-QDA get to those of lazy-ER. We measure this closeness in terms of the F-measure where the results of lazy-ER are treated as the ground truth.

Figure 12 shows that our tests of CC^+ and CC^- work very well, as lazy-QDA reaches very high F-measure values. Note that even when F-measure value drops to 0.95, lazy-QDA reaches about the same quality as lazy-ER in Figure 11 – since they both make errors, some of which are different.

6.2 Hotels Dataset Experiments

In this section, we run several queries on a real hotels dataset, which is larger than the Google Scholar dataset used in the previous section. This dataset includes hotels information (e.g., hotel-id, hotel-name, hotel-address, hotel-city, hotel-country, hotel-stars, hotel-price, etc.). It contains $184,169$ hotels where almost 40% are duplicates.

We use min-hashing [22] to generate a signature for each record (i.e., an array of integers where each integer is generated by applying a random hash function to the hotel-name of the record). Afterwards, we use locality-sensitive hashing [17] to place records with high similarity into into $1,000$ big blocks. Next, we apply the same blocking technique used in the previous section to further partition these big blocks. That is, we partition the records in each big block into smaller blocks based on the first two letters and the last two letters of the hotel's name. Thus, if the names of two hotels in one big block match in either the first or last two letters then they are put in the same small block.

We implemented a pairwise resolve function which operates on two records to decide whether they are duplicates. It uses Soft-TF-IDF to compare the names of hotels.

We can classify the different queries used in these tests into three different classes.

1) *Class one – Inexpensive good hotels in the US.* Queries in this class consist of the three predicates $p_1 : price \leq t_1$, $p_2 : stars \geq t_2$, and $p_3 : country =$ 'US'. Thus, these queries consist of three triples: an in-preserving triple $\tau_1 = (price \leq t_1, \texttt{MIN}, price)$, an in-preserving triple $\tau_2 = (stars \geq t_2, \texttt{MAX}, stars)$, and an in-preserving triple $\tau_3 = (country = \text{'US'}, \texttt{EXEMPLAR}, country)$. From Table 2, we can see that the resulting combination $\tau_1 \wedge \tau_2 \wedge \tau_3$ is in-preserving.
2) *Class two – Overpriced hotels.* Queries in this class consist of the two predicates $p_1 : price \geq t_1$ and $p_2 : stars \leq t_2$. Hence, such queries consist of two triples: an out-preserving triple $\tau_1 = (price \geq t_1, \texttt{MIN}, price)$ and an out-preserving triple $\tau_2 = (stars \leq t_2, \texttt{MAX}, stars)$. From Table 2, we can see that the resulting combination $\tau_1 \wedge \tau_2$ is out-preserving.
3) *Class three – Poor quality hotels.* Queries in this class consist of the two predicates $p_1 : stars \leq t_1$ and $p_2 : country = t_2$. Therefore, these queries consist of two triples: an out-preserving triple $\tau_1 = (stars \leq t_1, \texttt{MAX}, stars)$ and an in-preserving triple $\tau_2 = (country = t_2, \texttt{EXEMPLAR}, country)$. From Table 2, we can see that the resulting combination $\tau_1 \wedge \tau_2$ is neither in- nor out-preserving.

Experiment 10 (In-preserving Triples). In this test, we compare eager-QDA (using the representative answer semantics) versus eager-ER. This test shows the effects of testing for vestigiality by using four queries that belong to *class one*. Figure 13 plots the number of resolves invoked by eager-QDA and eager-ER for all four queries.

As expected, eager-QDA outperforms eager-ER since eager-QDA is aware of the query thresholds while eager-ER is not. This thresholds awareness gives eager-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 13, when the range of the *stars rating* is large (as in *Q1* and *Q2*), we can observe that when the *price* increases the number of calls to resolve decreases. This is because the number of records which satisfy the query increases and thus many records are labeled `in` causing the edges connected to them to be vestigial.

However, when the range of the *stars rating* is not large (as in *Q3* and *Q4*), we can see that when the *price* increases the number of calls to resolve neither increases nor decreases. This is because the majority of records are labeled `maybe` and there are many potential cliques that can satisfy the query.

Experiment 11 (Out-preserving Triples). This test uses two queries that belong to *class two* to compare eager-QDA (using representative answer semantics) and eager-ER. Figure 14 shows the number of resolves invoked by eager-QDA and eager-ER for the two queries.

Eager-QDA is superior to eager-ER for both queries. However, herein, eager-QDA issues a higher number of resolves compared to eager-QDA in the previous test. This is because, here, eager-QDA can only exploit the out-preserving property to remove records from the result-set. Recall that, 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 duplicates.

Note that the curves of *Q5* and *Q6* look alike since the two predicates are opposites. That is, when p_1 is satisfied, p_2 is not, and vice versa. For example, in *Q5*, there are not many hotels with *price* ≥ 200 and *stars* ≤ 1; thus, many nodes are labeled out and the edges connecting these nodes are vestigial. In contrast, in *Q6*, there are many hotels with *price* ≥ 200 and *stars* ≤ 3; however, there are not many potential cliques that can remove them from the answer. As a result, many of the edges connected to these nodes are vestigial.

Experiment 12 (Neither in-/out-preserving Triples). This experiment uses two queries that belong to *class three* to compare eager-QDA vs. eager-ER. Figure 15 demonstrates the number of calls to resolve by both eager-QDA and eager-ER for these two queries.

As shown in Figure 15, eager-QDA outperforms eager-ER for both queries. Note that the number of resolves called in this test is relative to the number of hotels that are located in the *country*. That is, the more hotels in the country, the larger the number of potential cliques that may alter the answer, and thus the more resolves that need to be called.

7 Conclusions and Future Work

In this paper, we have studied the query-driven ER problem in which data is cleaned "on-the-fly" in the context of a selection query. We have developed QDA, which efficiently issues the minimal number of cleaning steps needed to accurately answer a given selection query. We formalized the problem of query-driven ER and showed empirically how certain cleaning steps can be pruned. This research opens several interesting directions for future investigation (e.g., developing solutions for efficient maintenance of a database state for subsequent querying).

References

[1] http://web.cs.ucla.edu/~palsberg/h-number.html. [Online; accessed 30-June-2016].
[2] H. Altwaijry et al. Query-driven approach to entity resolution. *VLDB*, 2013.
[3] H. Altwaijry et al. Query: a framework for integrating entity resolution with query processing. *VLDB*, 2015.
[4] R. Ananthakrishna et al. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, 2002.
[5] N. Bansal et al. Correlation clustering. *Machine Learning*, 2004.
[6] O. Benjelloun et al. Swoosh: a generic approach to entity resolution. *VLDB J.*, 2009.
[7] I. Bhattacharya et al. Query-time entity resolution. *JAIR*, 2007.
[8] M. Bilenko et al. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *ICDM*, 2005.
[9] Z. Chen et al. Adaptive graphical approach to entity resolution. In *JCDL*, 2007.
[10] Z. Chen et al. Exploiting context analysis for combining multiple entity resolution systems. In *SIGMOD*, 2009.
[11] W. Cohen et al. A comparison of string metrics for matching names and records. In *IIWeb*, 2003.
[12] X. Dong et al. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.
[13] E. Elmacioglu et al. Web based linkage. In *WIDM*, 2007.
[14] A. K. Elmagarmid et al. Duplicate record detection: A survey. *TKDE*, 2007.
[15] W. Fan et al. Reasoning about record matching rules. *VLDB*, 2009.
[16] I. P. Fellegi et al. A theory for record linkage. *JASA*, 1969.
[17] A. Gionis et al. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.
[18] O. Hassanzadeh et al. Framework for evaluating clustering algorithms in duplicate detection. *VLDB*, 2009.
[19] M. A. Hernández et al. The merge/purge problem for large databases. In *SIGMOD Record*, 1995.
[20] M. A. Hernández et al. Real-world data is dirty: Data cleansing and the merge/purge problem. *DMKD*, 1998.
[21] T. N. Herzog et al. *Data quality and record linkage techniques*. Springer Science & Business Media, 2007.
[22] P. Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 2001.
[23] E. Ioannou et al. On-the-fly entity-aware query processing in the presence of linkage. *VLDB*, 2010.
[24] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *JASA*, 1989.
[25] D. V. Kalashnikov et al. Domain-independent data cleaning via analysis of entity-relationship graph. *TODS*, 2006.
[26] P. H. Kanani et al. Improving author coreference by resource-bounded information gathering from the web. In *IJCAI*, 2007.
[27] H. Kellerer et al. Two linear approximation algorithms for the subset-sum problem. *EJOR*, 2000.
[28] A. McCallum et al. Efficient clustering of high-dimensional data sets with application to reference matching. In *SIGKDD*, 2000.
[29] H. B. Newcombe et al. Automatic linkage of vital records computers can be used to extract" follow-up" statistics of families from files of routine records. *Science*, 1959.
[30] R. Nuray-Turan et al. Exploiting web querying for web people search. *TODS*, 2012.
[31] W. Su et al. Record matching over query results from multiple web databases. *TKDE*, 2010.
[32] J. Wang et al. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, 2014.
[33] S. E. Whang et al. Entity resolution with evolving rules. *VLDB*, 2010.
[34] M. Yakout et al. Behavior based record linkage. *VLDB*, 2010.
[35] L. Zhang et al. A unified framework for context assisted face clustering. In *ICMR*, 2013.

Hotham Altwaijry is an Assistant Research Professor in the Center of Complex Engineering Systems (CCES) at King Abdulaziz City for Science and Technology (KACST) and MIT. He received his PhD degree in Computer Science from UC Irvine in 2015. He has received several scholarships, awards, and honors, including a KACST's Graduate Studies Scholarship. His primary research interests are in the areas of: data management, data quality, data cleaning, entity resolution, and big data.

Dmitri V. Kalashnikov is a Senior Inventive Scientist at AT&T Labs Research. Prior to that, he had worked for 12 years in academia as an Associate Adjunct Professor at UC Irvine, 2003–2015. He received his PhD degree in Computer Science from Purdue University in 2003. He received his diploma in Applied Mathematics and Computer Science from Moscow State University, Russia in 1999, graduating summa cum laude. His general research interests include data management, databases, and data mining. Currently, he specializes in the areas of data quality and data integration for big data. In the past, he has also contributed to the areas of spatial databases, moving-object databases, probabilistic databases, and real-time situational awareness. He has received several scholarships, awards, and honors, including an Intel Fellowship, Intel Scholarship, and a Best Paper award at ACM ICMR 2013 conference.

Sharad Mehrotra received the PhD degree in computer science from the University of Texas at Austin in 1993. He is currently a professor in the Department of Computer Science at the University of California, Irvine (UCI) and the director of the Center for Emergency Response Technologies. Previously, he was a professor at the University of Illinois at UrbanaChampaign (UIUC). He has received numerous awards and honors, including SIGMOD Best Paper award 2001, DASFAA Best Paper award 2004, and CAREER award 1998 from the US National Science Foundation (NSF). His primary research interests are in the area of database management, distributed systems, and data analysis. He is a member of the IEEE.