The VLDB Journal (2014) 23:77–102
DOI 10.1007/s00778-013-0315-0

REGULAR PAPER

Incremental entity resolution on rules and data

Steven Euijong Whang · Hector Garcia-Molina

Received: 20 August 2012 / Revised: 30 March 2013 / Accepted: 3 April 2013 / Published online: 17 April 2013

Abstract Entity resolution (ER) identifies database records that refer to the same real-world entity. In practice, ER is not a one-time process, but is constantly improved as the data, schema and application are better understood. We first address the problem of keeping the ER result up-to-date when the ER logic or data "evolve" frequently. A naïve approach that re-runs ER from scratch may not be tolerable for resolving large datasets. This paper investigates when and how we can instead exploit previous "materialized" ER results to save redundant work with evolved logic and data. We introduce algorithm properties that facilitate evolution, and we propose efficient rule and data evolution techniques for three ER models: match-based clustering (records are clustered based on Boolean matching information), distance-based clustering (records are clustered based on relative distances), and pairs ER (the pairs of matching records are identified). Using real datasets, we illustrate the cost of materializations and the potential gains of evolution over the naïve approach.

Keywords Entity resolution · Rule evolution · Data evolution · Data cleaning

S. E. Whang (✉) · H. Garcia-Molina
Computer Science Department, Stanford University,
Stanford, CA 94305, USA
e-mail: swhang@cs.stanford.edu

H. Garcia-Molina
e-mail: hector@cs.stanford.edu

Present address:
S. E. Whang
Google Inc., Mountain View, CA, USA

1 Introduction

Entity resolution [8,10,19,31] (also known as record linkage or deduplication) is the process of identifying records that represent the same real-world entity. For example, two companies that merge may want to combine their customer records. In such a case, the same customer may be represented by multiple records, so these matching records must be identified and combined (into what we will call a cluster). This ER process is often extremely expensive due to very large datasets and complex logic that decides when records represent the same entity.

In practice, an entity resolution (ER) result is not produced once, but is constantly improved based on better understandings of the data, the schema, and the logic that examines and compares records. For example, suppose a database systems and applications vendor acquire a computer hardware company. The type of products the company purchases changes significantly, to include computer chips, cooling fans, and so on. In this case, the logic for resolving the products purchased from suppliers can change significantly. While any new data can be resolved using the new logic, the existing resolved data need to be updated as well. This update can be time-consuming and redundant, especially if mergers (of suppliers, vendors, acquired companies), changes in tax codes, introduction of new products, changes to business strategy, changes to compliance regulations, and so on, are frequent.

In particular, here we focus on (1) changes to the logic that decides when records represent the same entity and (2) new data to resolve. Figure 1 shows our general approach for incremental ER. Suppose that we have computed the ER result E_1 using the ER logic on a set of input records S. Given new information in the form of rules or data, we would like to produce the new ER result E_2 efficiently. Instead of re-running ER from scratch and producing E_2 from S (fol-

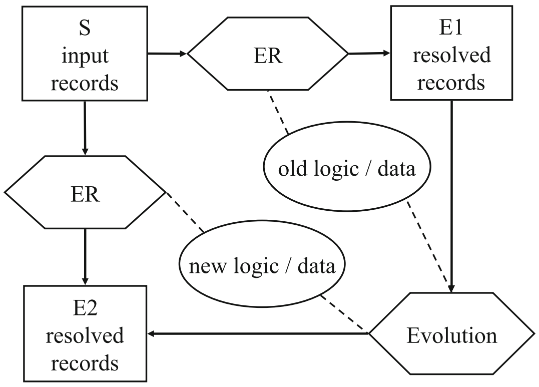

Fig. 1 Incremental ER

Table 1 Records to resolve

Record	Name	Zip	Phone
r_1	*John*	54321	123-4567
r_2	*John*	54321	987-6543
r_3	*John*	11111	987-6543
r_4	*Bob*	*null*	121-1212

Table 2 Evolving from rule B_1 to rule B_3

Comparison rule	Definition
B_1	p_{name}
B_2	$p_{name} \wedge p_{zip}$
B_3	$p_{name} \wedge p_{phone}$

lowing the vertical arrow from the top left box to the bottom left box), our approach is to efficiently derive E_2 from E_1 using evolution techniques (following the top right box to the bottom left box).

We first solve the problem of incremental ER on evolving logic. Initially, we start with a set of records S and then produce a first ER result E_1 based on S and a rule B_1, which determines if two records represent the same entity. Some time later, rule B_1 is improved yielding rule B_2, so we need to compute a new ER result E_2 based on S and B_2. The process continues with new rules B_3, B_4, and so on.

To motivate and explain our approach focusing on rules, consider the following example. Our initial set of people records S is shown in Table 1. The first rule B_1 (see Table 2) says that two records match (represent the same real-world entity) if predicate p_{name} evaluates to true. Predicates can in general be quite complex, but for this example, assume that predicates simply perform an equality check. The ER algorithm calls on B_1 to compare records and groups together records with name "John," producing the result $\{\{r_1, r_2, r_3\}, \{r_4\}\}$. (As we will see, there are different types of ER algorithms, but in this simple case, most would return this same result.)

Next, say users are not satisfied with this result, so a data administrator decides to refine B_1 by adding a predicate that checks zip codes. Thus, the new rule is B_2 shown in Table 2. The naïve option is to run the same ER algorithm with rule B_2 on set S to obtain the partition $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$. (Only records r_1 and r_2 have the same name and same zip code.) This process repeats much unnecessary work: For instance, we would need to compare r_1 with r_4 to see whether they match on name and zip code, but we already know from the first run that they do not match on name (B_1), so they cannot match under B_2.

Because the new rule B_2 is stricter than B_1 (we define this term precisely later on), we can actually start the second ER from the first result $\{\{r_1, r_2, r_3\}, \{r_4\}\}$. That is, we only need to check each cluster separately and see if it needs to split. In our example, we find that r_3 does not match the other records in its cluster, so we arrive at $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$. This approach only works if the ER algorithm satisfies certain properties and B_2 is stricter than B_1. If B_2 is not stricter and the ER algorithm satisfies different properties, there are other incremental techniques we can apply based on materialization (explained below). Our goal in this paper is to explore these options: Under what conditions and for what ER algorithms are incremental approaches *feasible*? And in what scenarios are the savings over the naïve approach significant?

In addition, we study a complementary technique: *materialize* auxiliary results during one ER run, in order to improve the performance of future ER runs. To illustrate, say that when we process $B_2 = p_{name} \wedge p_{zip}$, we concurrently produce the results for each predicate individually. That is, we compute three separate partitions, one for the full B_2, one for rule p_{name}, and one for rule p_{zip}. The result for p_{name} is the same $\{\{r_1, r_2, r_3\}, \{r_4\}\}$ seen earlier. For p_{zip}, it is $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$. As we will see later, the cost of computing the two extra materializations can be significantly lower than running the ER algorithm three times, as a lot of the work can be shared among the runs.

The materializations pay off when rule B_2 evolves into a related rule that is not quite stricter. For example, say that B_2 evolves into $B_3 = p_{name} \wedge p_{phone}$, where p_{phone} checks for matching phone numbers. In this case, B_3 is not stricter than B_2, so we cannot start from the B_2 result. However, we can start from the p_{name} result, since B_3 is stricter than p_{name}. Thus, we independently examine each cluster in $\{\{r_1, r_2, r_3\}, \{r_4\}\}$, splitting the first cluster because r_1 has a different phone number. The final result is $\{\{r_1\}, \{r_2, r_3\}, \{r_4\}\}$. Clearly, materialization of partial results may or may not pay off, just like materialized views and indexes may or may not help. Our objective here is, again, to study when is materialization *feasible* and to illustrate scenarios where it can pay off.

The second problem we solve is incremental ER on new data. This time, the rule remains the same, but new records may now have to be resolved in addition to the original records. For example, after resolving the four records in Table 1, we may have to resolve two more records r_5 and r_6. While resolving the additional records, we would like to avoid redundant record comparisons as many as possible.

In summary, our contributions in this paper are as follows:

- We define an ER model (Sect. 2) that clusters records and formalize evolution for Boolean comparison rules. We identify two desirable properties of ER algorithms (rule monotonic and context free) that enable efficient evolution. We also contrast these properties to two properties mentioned in the literature (order independent and incremental).
- We propose efficient rule and data evolution techniques that use one or more of the four properties (Sects. 3, 4). We believe that our results can be a useful guide for ER algorithm designers: If they need to handle evolving rules efficiently, they may want to build algorithms that have at least some of the properties we present.
- We explore two variations of the evolution problem:
  - The comparison rule is a distance function instead of a Boolean function.
  - An ER algorithm returns pairs of matching records instead of a partition.

  We provide rule and data evolution techniques analogous to those for clustering-based ER using Boolean comparison rules (Sect. 5).
- We categorize a number of existing ER algorithms based on the properties they satisfy (Sect. 6).
- We experimentally evaluate (Sect. 7) the rule and data evolution algorithms for various clustering-based ER algorithms using actual comparison shopping data from Yahoo! Shopping and hotel information from Yahoo! Travel. Our results show scenarios where rule evolution can be faster than the naïve approach by up to several orders of magnitude. We also illustrate the time and space cost of materializing partial results and argue that these costs can be amortized with a small number of future evolutions. Finally, we also experiment with ER algorithms that do not satisfy our properties and show that if one is willing to sacrifice accuracy, one can still use our rule evolution techniques.

2 Model and properties

Throughout this paper, we consider three ER models: match-based clustering , distance-based clustering, and pairs ER. In this section, we focus on the match-based clustering model, which uses a Boolean comparison rule for resolving records. We formalize the ER model and discuss two important properties for ER algorithms that can significantly enhance the runtime of evolution. We compare the two properties with existing properties for ER algorithms in the literature. In Sect. 5, we show that not much changes for the corresponding properties in distance-based clustering and pairs ER.

2.1 Match-based clustering model

We define a Boolean comparison rule B as a function that takes two records and returns `true` or `false`. We assume that B is commutative, i.e., $\forall r_i, r_j, B(r_i, r_j) = B(r_j, r_i)$.

Suppose we are given a set of records $S = \{r_1, \ldots, r_n\}$. An ER algorithm receives as inputs a partition P_i of S and a Boolean comparison rule B and returns another partition P_o of S. A partition of S is defined as a set of clusters $P = \{c_1, \ldots, c_m\}$ such that $c_1 \cup \ldots \cup c_m = S$ and $\forall c_i, c_j \in P$ where $i \neq j, c_i \cap c_j = \emptyset$.

We require the input to be a partition of S so that we may also run ER on the output of a previous ER result. In our motivating example in Sect. 1, the input was a set of records $S = \{r_1, r_2, r_3, r_4\}$, which can be viewed as a partition of singletons $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}\}$, and the output using the comparison rule $B_2 = p_{name} \wedge p_{zip}$ was the partition $P_o = \{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$. If we run ER a second time on the ER output $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$, we may obtain the new output partition $P_o = \{\{r_1, r_2, r_3\}, \{r_4\}\}$ where the cluster $\{r_1, r_2\}$ accumulated enough information to match with the cluster $\{r_3\}$. An alternative ER model is to accept a set of records and return another set of merged records. However, we believe that using partitions of records is more convenient for our purposes without reducing expressiveness.

How exactly the ER algorithm uses B to derive the output partition P_o depends on the specific ER algorithm. The records are clustered based on the results of B when comparing records. In our motivating example (Sect. 1), all pairs of records that matched according to $B_2 = p_{name} \wedge p_{zip}$ were clustered together. Note that, in general, an ER algorithm may not cluster two records simply because they match according to B. For example, two records r and s may be in the same cluster $c \in P_o$ even if $B(r, s) =$ `false`. Or the two records could also be in two different clusters $c_i, c_j \in P_o (i \neq j)$ even if $B(r, s) =$ `true`.

We also allow input clusters to be un-merged as long as the final ER result is still a partition of the records in S. For example, given an input partition $\{\{r_1, r_2, r_3\}, \{r_4\}\}$, an output of an ER algorithm could be $\{\{r_1, r_2\}, \{r_3, r_4\}\}$ and not necessarily $\{\{r_1, r_2, r_3\}, \{r_4\}\}$ or $\{\{r_1, r_2, r_3, r_4\}\}$. Un-merging could occur when an ER algorithm decides that some records were incorrectly clustered [28].

Finally, we assume the ER algorithm to be *non-deterministic* in a sense that different partitions of S may be produced depending on the order of records processed or by some random factor (e.g., the ER algorithm could be a randomized algorithm). For example, a hierarchical clustering algorithm based on Boolean rules (see Sect. 6.1) may produce different partitions depending on which records are compared first. While the ER algorithm is non-deterministic, we assume the comparison rule itself to be deterministic, i.e., it always returns the same matching result for a given pair of records.

We now formally define a valid ER algorithm.

Definition 1 Given any input partition P_i of a set of records S and any Boolean comparison rule B, a *valid* ER algorithm E non-deterministically returns an ER result $E(P_i, B)$ that is also a partition P_o of S.

Note that Definition 1 covers deterministic ER algorithms as well.

We denote all the possible partitions that can be produced by the ER algorithm E as $\bar{E}(P_i, B)$, which is a set of partitions of S. Hence, $E(P_i, B)$ is always one of the partitions in $\bar{E}(P_i, B)$. For example, given $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$, $\bar{E}(P_i, B)$ could be $\{\{\{r_1, r_2\}, \{r_3\}\}, \{\{r_1\}, \{r_2, r_3\}\}\}$ while $E(P_i, B) = \{\{r_1, r_2\}, \{r_3\}\}$.

An important concept used in evolution is the relative strictness between comparison rules:

Definition 2 A Boolean comparison rule B_1 is *stricter than* another rule B_2 (denoted as $B_1 \leq B_2$) if $\forall r_i, r_j$, $B_1(r_i, r_j) = \texttt{true}$ implies $B_2(r_i, r_j) = \texttt{true}$.

For example, a comparison rule B_1 that compares the string distance of two names and returns `true` when the distance is lower than 5 is stricter than a comparison rule B_2 that uses a higher threshold of, say, 10. As another example, a comparison rule B_1 that checks whether the names and addresses are same is stricter than another rule B_2 that only checks whether the names are same.

Our incremental ER approach is based on Boolean match functions or distance functions for pairs of records. Machine learning (ML), probabilistic modeling, and graph-based approaches can be used to develop and refine these functions. For instance, the developer of the ER algorithm can use ML and training data to determine what features and thresholds to use for deciding if records match, or in determining the distance between records. If an improved training set is obtained, then ML can be used to obtain new match or distance functions, and our strategy can be used to incrementally obtain a new resolution. However, if the resulting ML-based ER algorithm cannot be described via match or distance functions, then our techniques are not applicable. In this later case, we suspect that incremental resolution will be extremely difficult.

When applying the incremental ER techniques, the application developer must analyze her ER algorithm and check if it satisfies certain properties, which we define from Sect. 2.2. For example, Collective ER techniques [4] that use similarity functions must satisfy the properties in Sect. 5.1.2 to use our framework.

2.2 Properties

We introduce two important properties for ER algorithms – *rule monotonic* and *context free* – that enable efficient evolution for match-based clustering.

2.2.1 Rule monotonic

Before defining the rule monotonic property, we first define the notion of refinement between partitions.

Definition 3 A partition P_1 of a set S *refines* another partition P_2 of S (denoted as $P_1 \leq P_2$) if $\forall c_1 \in P_1, \exists c_2 \in P_2$ s.t. $c_1 \subseteq c_2$.

For example, given the partitions $P_1 = \{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$ and $P_2 = \{\{r_1, r_2, r_3\}, \{r_4\}\}$, $P_1 \leq P_2$ because $\{r_1, r_2\}$ and $\{r_3\}$ are subsets of $\{r_1, r_2, r_3\}$ while $\{r_4\}$ is a subset of $\{r_4\}$.

We now define the rule monotonic property, which guarantees that the stricter the comparison rule, the more refined the ER result.

Definition 4 An ER algorithm is *rule monotonic* ($\mathcal{RM}$) if, for any three partitions P, P_o^1, P_o^2 and two comparison rules B_1 and B_2 such that

- $B_1 \leq B_2$ and
- $P_o^1 \in \bar{E}(P, B_1)$ and
- $P_o^2 \in \bar{E}(P, B_2)$

then $P_o^1 \leq P_o^2$.

An ER algorithm satisfying $\mathcal{RM}$ guarantees that, if the comparison rule B_1 is stricter than B_2, the ER result produced with B_1 refines the ER result produced with B_2. For example, suppose that $P = \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}\}$, $B_1 \leq B_2$, and $E(P_i, B_1) = \{\{r_1, r_2, r_3\}, \{r_4\}\}$. If the ER algorithm is $\mathcal{RM}$, $E(P_i, B_2)$ can only return $\{\{r_1, r_2, r_3\}, \{r_4\}\}$ or $\{\{r_1, r_2, r_3, r_4\}\}$.

2.2.2 Context free

The second property, context free, tells us when a subset of P_i (called P, which is a partition of a subset of S) can be processed "in isolation" from the rest of the clusters. (For clarification, the second condition says that none of the records in P can match with any of the records in $P_i - P$).

Definition 5 An ER algorithm is *context free* ($\mathcal{CF}$) if for any four partitions P, P_i, P_o^1, P_o^2 and a comparison rule B such that

- $P \subseteq P_i$ and
- $\forall P_o \in \bar{E}(P_i, B)$, $P_o \leq \{\bigcup_{c \in P} c, \bigcup_{c \in P_i - P} c\}$ and
- $P_o^1 \in \bar{E}(P, B)$ and
- $P_o^2 \in \bar{E}(P_i - P, B)$

then $P_o^1 \cup P_o^2 \in \bar{E}(P_i, B)$.

Suppose that we are resolving $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}\}$ with the knowledge that no clusters in $P = \{\{r_1\}, \{r_2\}\}$ will merge with any of the clusters in $P_i - P = \{\{r_3\}, \{r_4\}\}$. Then, for any $P_o \in \bar{E}(P_i, B)$, $P_o \leq \{\{r_1, r_2\}, \{r_3, r_4\}\}$. In this case, an ER algorithm that is $\mathcal{CF}$ can resolve $\{\{r_1\}, \{r_2\}\}$ independently from $\{\{r_3\}, \{r_4\}\}$, and there exists an ER result of P_i that is the same as the union of the ER results of $\{\{r_1\}, \{r_2\}\}$ and $\{\{r_3\}, \{r_4\}\}$.

2.3 Existing ER properties

To get a better understanding of $\mathcal{RM}$ and $\mathcal{CF}$, we compare them to two existing properties in the literature: incremental and order independent.

An ER algorithm is incremental [18,19] if it can resolve one record at a time. We define a more generalized version of the incremental property for our ER model where any subsets of clusters in P_i can be resolved at a time.

Definition 6 An ER algorithm is *general incremental* ($\mathcal{GI}$) if for any four partitions P, P_i, P_o^1, P_o^2, and a comparison rule B such that

- $P \subseteq P_i$ and
- $P_o^1 \in \bar{E}(P, B)$ and
- $P_o^2 \in \bar{E}(P_o^1 \cup (P_i - P), B)$

then $P_o^2 \in \bar{E}(P_i, B)$.

For example, suppose we have $P = \{\{r_1\}, \{r_2\}\}$, $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$, and $P_o^1 = \{\{r_1, r_2\}\}$. That is, we have already resolved P into the result P_o^1. We can then add to P_o^1 the remaining cluster $\{r_3\}$ and resolve all the clusters together. The result is as if we had resolved everything from scratch (i.e., from P_i). Presumably, the former way (incremental) will be more efficient than the latter.

The $\mathcal{GI}$ property is similar to the $\mathcal{CF}$ property, but also different in a number of ways. First, $\mathcal{GI}$ and $\mathcal{CF}$ are similar in a sense that they use two subsets of P_i: P and $P_i - P$. However, under $\mathcal{GI}$, $P_i - P$ is not resolved until P has been resolved. Also, $\mathcal{GI}$ does not assume P and $P_i - P$ to be independent (i.e., a cluster in P may merge with a cluster in $P_i - P$).

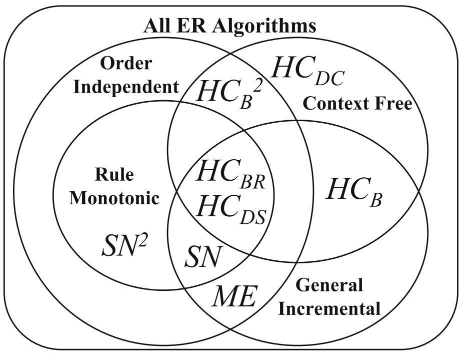

Fig. 2 ER Algorithms satisfying properties

We now explore the second property in the literature. An ER algorithm is order independent ($\mathcal{OI}$) [19] if the ER result is same regardless of the order of the records processed. That is, for any input partition P_i and comparison rule B, $\bar{E}(P_i, B)$ is a singleton (i.e., $\bar{E}(P_i, B)$ contains exactly one partition of S). An order independent ER algorithm is also deterministic.

The venn diagram in Fig. 2 shows various ER algorithms that satisfy one or more of the four properties. The details on how the properties relate to each other, the definitions of the ER algorithms, and the proofs on why each ER algorithm satisfies its properties are later explained in Sect. 6. For now, we briefly describe three basic ER algorithms: HC_B, SN, and ME. The hierarchical clustering algorithm (HC_B) combines matching pairs of clusters in any order until now clusters match with each other. The sorted neighbor algorithm (SN) sorts the input records by a key and then slides a fixed-sized window while matching the records within the same window. Finally, the Monge–Elkan algorithm (ME) maintains a fixed-length queue containing clusters and compares each new record with the clusters in the queue.

In the following sections, we propose various *incremental ER algorithms*. The input of incremental rule evolution consists of an ER algorithm E, an input partition P_i, an old rule B_1, and a new rule B_2. The input of incremental data evolution consists of an ER algorithm E, a rule B, an input partition P_i, and another partition P_i' that contains new records to resolve with P_i. The output of both types of incremental evolution is a partition of records. We explore incremental ER algorithms for rule evolution in Sect. 3 and data evolution in Sect. 4.

3 Rule evolution

As described in Sect. 1, rule evolution occurs when the Boolean comparison rule of an ER algorithm improves. In this section, we present rule evolution techniques that incrementally update ER results. We first explain how to materialize previous ER results for fast rule evolution. Next, we present efficient rule evolution techniques for ER algorithms using the properties in Sect. 2. Finally, we discuss more advanced materialization strategies that can further improve rule evolution efficiency.

3.1 Materialization

To improve our chances that we can efficiently compute a new ER result with rule B_2, when we compute earlier results we can materialize results that involve predicates likely to be in B_2. In particular, let us assume that rules are Boolean expressions of smaller binary predicates. For example, a rule that compares the names and addresses of two people can be defined as $p_{name} \wedge p_{address}$ where p_{name} could be a function that compares the names of two people while the predicate $p_{address}$ could compare the street addresses and apartment numbers of two people. In general, a predicate can be any function that compares an arbitrary number of attributes. We assume that all predicates are commutative and (without loss of generality) all rules are in conjunctive normal form (CNF). For example, the rule $B = p_1 \wedge p_2 \wedge (p_3 \vee p_4)$ is in CNF and has three conjuncts p_1, p_2, and $p_3 \vee p_4$.

When we compute an earlier result $E(P_i, B_1)$ where say $B_1 = p_1 \wedge p_2 \wedge p_3$, we can also materialize results such as $E(P_i, p_1)$, $E(P_i, p_2)$, $E(P_i, p_1 \wedge p_2)$, and so on. The most useful materializations will be those that can help us later with $E(P_i, B_2)$. (See Sect. 3.3.) For concreteness, here we will assume that we materialize *all* conjuncts of B_1 (in our example, $E(P_i, p_1)$, $E(P_i, p_2)$, and $E(P_i, p_3)$).

Instead of serially materializing each conjunct, however, we can amortize the common costs by materializing different conjuncts in a concurrent fashion. For example, parsing and initializing the records can be done once during the entire materialization. More operations can be amortized depending on the given ER algorithm. For example, when materializing conjuncts using an ER algorithm that always sorts its records before resolving them, the records only need to be sorted once for all materializations. In Sect. 7.5, we show that amortizing common operations can significantly reduce the time overhead of materializing conjuncts. A partition of the records in S can be stored compactly in various ways. One approach is to store sets of record IDs in a set where each inner set represents a cluster of records. A possibly more space-efficient technique is to maintain an array A of records (where the ID is used as the index) where each cell contains the cluster ID. For example, if r_5 is in the second cluster, then $A[5] = 2$. If there are only a few clusters, we only need a small number of bits for saving each cluster ID. For example, if there are only 8 clusters, then each entry in A only takes 3 bits of space.

Algorithm 1 Rule evolution given $\mathcal{RM}$ and $\mathcal{CF}$

```
1: input: The input partition $P_i$, the comparison rules $B_1$, $B_2$, the ER result for each conjunct of $B_1$, the hash table $H$ containing ER materializations of conjuncts in $B_1$
2: output: The output partition $P_o \in \bar{E}(P_i, B_2)$
3: Partition $M \leftarrow \bigwedge_{conj \in Conj(B_1) \cap Conj(B_2)} H(conj)$
4: return $\bigcup_{c \in M} E(\{c' \in P_i | c' \subseteq c\}, B_2)$
```

3.2 Rule evolution

We provide efficient rule evolution techniques for ER algorithms using the properties. Our first algorithm supports ER algorithms that are $\mathcal{RM}$ and $\mathcal{CF}$. As we will see, rule evolution can still be efficient for ER algorithms that are only $\mathcal{RM}$. Our second algorithm supports ER algorithms that are $\mathcal{GI}$. Before running the rule evolution algorithms, we materialize ER results for conjuncts of the old comparison rule B_1 by storing a partition of the input records S (i.e., the ER result) for each conjunct in B_1 (see Sect. 3.3 for possible optimizations). In general, we suspect that (although we will not explicitly show) the number of properties satisfied by the ER algorithm is correlated with better runtime performance.

To explain our rule evolution algorithms, we review a basic operation on partitions. The *meet* of two partitions P_1 and P_2 (denoted as $P_1 \wedge P_2$) returns a new partition of S whose members are the non-empty intersections of the clusters of P_1 with those of P_2. For example, given the partitions $P_1 = \{\{r_1, r_2, r_3\}, \{r_4\}\}$ and $P_2 = \{\{r_1\}, \{r_2, r_3, r_4\}\}$, the meet of P_1 and P_2 becomes $\{\{r_1\}, \{r_2, r_3\}, \{r_4\}\}$ since r_2 and r_3 are clustered in both partitions. We also show the following lemma regarding the meet operation, which is a standard result in set theory. Proofs for this result and subsequent ones can be found in Appendix 10.

Lemma 1 *If* $\forall i$, $P \leq P_i$, *then* $P \leq \bigwedge P_i$.

3.2.1 ER algorithms satisfying $\mathcal{RM}$ and $\mathcal{CF}$

Algorithm 1 performs rule evolution for ER algorithms that are both $\mathcal{RM}$ and $\mathcal{CF}$. (We define real ER algorithms that can be plugged into Algorithm 1 in Sect. 6.) The input requires the input partition P_i, the old, and new comparison rules (B_1 and B_2, respectively), and a hash table H that contains the materialized ER results for the conjuncts of B_1. The conjuncts of a comparison rule B are denoted as $Conj(B)$. For simplicity, we assume that B_1 and B_2 share at least one conjunct. Step 3 exploits the $\mathcal{RM}$ property and meets the partitions of the common conjuncts between B_1 and B_2. For example, suppose that we have $B_1 = p_1 \wedge p_2 \wedge p_3$ and $B_2 = p_1 \wedge p_2 \wedge p_4$. Given $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}\}$, say we also have the materialized ER results $E(P_i, p_1) = \{\{r_1, r_2, r_3\}, \{r_4\}\}$ and $E(P_i, p_2) = E(P_i, p_3) = \{\{r_1\}, \{r_2, r_3, r_4\}\}$. Since the common conjuncts of B_1 and B_2 are p_1 and p_2, we generate the meet of $E(P_i, p_1)$ and $E(P_i, p_2)$ as $M = \{\{r_1\}, \{r_2, r_3\}, \{r_4\}\}$. By $\mathcal{RM}$, we know that $E(P_i, B_2)$ refines M because B_2 is stricter than both p_1 and p_2. That is, each cluster in the new ER result is contained in exactly one cluster in the meet M. Step 4 then exploits the $\mathcal{CF}$ property to resolve for each cluster c of M, the clusters in P_i that are subsets of c (i.e., $\{c' \in P_i | c' \subseteq c\}$). Since the clusters in different $\{c' \in P_i | c' \subseteq c\}$'s do not merge with each other, each $\{c' \in P_i | c' \subseteq c\}$ can be resolved independently. As a result, we can return $\{\{r_1\}\} \cup E(\{\{r_2\}, \{r_3\}\}, B_2) \cup \{\{r_4\}\}$ as the new ER result of B_2.

In order to prove that Algorithm 1 is correct, we first prove three lemmas below. For simplicity, we denote $\{c' \in P_i | c' \subseteq c\}$ as $\text{IN}(P_i, c)$. For a set of clusters Q, we define $\text{IN}(P_i, Q) = \bigcup_{c \in Q} \text{IN}(P_i, c)$.

Lemma 2 *For an $\mathcal{RM}$ algorithm, $\forall P_o \in \bar{E}(P_i, B_2)$, $P_o \leq M$.*

Lemma 3 *Suppose we have an algorithm that is $\mathcal{RM}$ and $\mathcal{CF}$, an initial partition P_i, and two rules B_1, B_2 with the conjuncts $Conj(B_1)$, $Conj(B_2)$, respectively. Let $M = \bigwedge_{conj \in Conj(B_1) \cap Conj(B_2)} E(P_i, conj)$. For any $W \subseteq M$, $E(IN(P_i, W), B_2) \leq W$.*

Lemma 4 *Given the same setup as in Lemma 3, let $Y \subseteq M$ and $Z \subseteq M$ such that $Y \cap Z = \emptyset$ and $Y \cup Z = W$ (note: $W \subseteq M$). Let $Q = E(IN(P_i, W), B_2)$ (there is only one solution). Then $Q \leq \{\bigcup_{c \in Y} c, \bigcup_{c \in Z} c\}$.*

Proposition 1 *Algorithm 1 correctly returns a partition $P_o \in \bar{E}(P_i, B_2)$.*

Proposition 2 *The complexity of Algorithm 1 is $O(c \times |S| + \frac{|S|^c}{|S|^{c-1}+z^c} \times g(\frac{|P_i| \times (|S|^{c-1}+z^c)}{|S|^c}, \frac{|S|}{|P_i|}))$ where S is the set of records in the input partition of records P_i, c is the number of common conjuncts between B_1 and B_2, z is the average cluster size for any partition produced by a conjunct, and $g(N, A)$ is the complexity of the ER algorithm E for an input partition containing N clusters with an average size of A records.*

While Algorithm 1 does not improve the complexity of the given ER algorithm E running without rule evolution, its runtime can be much faster in practice because the overhead for meeting partitions is not high (Step 3), and there can be large savings by running ER on small subsets of P_i (i.e., the $\{c' \in P_i | c' \subseteq c\}$'s) (Step 4) rather than on the entire partition P_i.

We can also prove that the space complexity of Algorithm 1 is $O(|S| + h(\frac{|P_i| \times (|S|^{c-1}+z^c)}{|S|^c}, \frac{|S|}{|P_i|}))$ using the notations in Proposition 2 and defining $h(N, A)$ as the space complexity of the ER algorithm E for an input partition containing N clusters with an average size of A records.

3.2.2 ER algorithms satisfying $\mathcal{RM}$ only

The rule evolution algorithm for ER algorithms that are only $\mathcal{RM}$ is identical to Algorithm 1 except for Step 4, where we can no longer process subsets of P_i independently. However, we can still run Step 4 efficiently using global information. Consider the sorted neighborhood ER algorithm (SN), which is defined in Sect. 6.1. The first step of SN is to move a sliding window on a sorted list of records, comparing records pairwise only within the same window of size W. (The second step is a transitive closure of all matching pairs.) In Step 4, we are able to resolve each $\{c' \in P_i | c' \subseteq c\} (c \in M)$ using the same window size W as long as we also use the global sort information of the records to make sure only the records that would have been in the same window during the original run of SN should be compared with each other. Suppose that we have $B_1 = p_{name} \wedge p_{zip}$, $B_2 = p_{name} \wedge p_{phone}$, and the initial set $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}, \{r_5\}\}$. We set the sort key to be the record ID (e.g., r_4 has the ID 4). As a result, the records are sorted into the list $[r_1, r_2, r_3, r_4, r_5]$. Using a window size of $W = 3$, suppose we materialize $E(P_i, p_{name}) = \{\{r_1, r_3, r_5\}, \{r_2\}, \{r_4\}\}$ because r_1 and r_3 matched when the window covered $[r_1, r_2, r_3]$ and r_3 and r_5 matched when the window covered $[r_3, r_4, r_5]$. The records r_1 and r_5 only match during the transitive closure in the second step of SN. The meet M in Algorithm 1 is also $\{\{r_1, r_3, r_5\}, \{r_2\}, \{r_4\}\}$ because there is only one common conjunct p_{name} between B_1 and B_2. Thus, we only need to resolve the set $\{r_1, r_3, r_5\}$ using B_2. However, we must be careful and should not simply run $E(\{r_1, r_3, r_5\}, B_2)$ using a sliding window of size 3. Instead, we must take into account the global ordering information and never compare r_1 and r_5, which were never in the same window. Thus, if $B_2(r_1, r_3) =$ `false`, $B_2(r_3, r_5) =$ `false`, and $B_2(r_1, r_5) =$ `true`, the correct ER result is that none of r_1, r_3, r_5 are clustered. While we need to use the global sort information of records, our rule evolution is still more efficient than re-running SN on the entire input P_i (see Sect. 7). In general, the rule evolution techniques for other ER algorithms that only satisfy $\mathcal{RM}$ may vary.

3.2.3 ER algorithms satisfying $\mathcal{GI}$

Algorithm 2 performs rule evolution for ER algorithms that satisfy the $\mathcal{GI}$ property. Algorithm 2 is identical to Algorithm 1 except for Step 4. Since the $\mathcal{RM}$ property is not satisfied anymore, we can no longer assume that the meet M is refined by the ER result of B_2. Hence, after each $\{c' \in P_i | c' \subseteq c\}$ is resolved, we need to run ER on the union of the results (i.e., the outermost ER operation in Step 4) to make sure we found all the matching records. The $\mathcal{GI}$ property guarantees that the output P_o is equivalent to a result

Algorithm 2 Rule evolution given $\mathcal{GI}$

```
1: input: The input partition $P_i$, the comparison rules $B_1$, $B_2$, the ER result for each conjunct of $B_1$, the hash table $H$ containing ER materializations of conjuncts in $B_1$
2: output: The output partition $P_o \in \bar{E}(P_i, B_2)$
3: Partition $M \leftarrow \bigwedge_{conj \in Conj(B_1) \cap Conj(B_2)} H(conj)$
4: return $E(\bigcup_{c \in M} E(\{c' \in P_i | c' \subseteq c\}, B_2), B_2)$
```

in $\bar{E}(P_i, B_2)$. Using the same example for Algorithm 1, we now return $E(\{\{r_1\}\} \cup E(\{r_2, r_3\}, B_2) \cup \{\{r_4\}\}, B_2)$.

There are two factors that make Algorithm 2 efficient for certain ER algorithms. First, each cluster in M is common to several ER results and thus contains records that are likely to be clustered. An ER algorithm may run faster by resolving clusters that are likely to match first. Second, there are fewer clusters for the outer E operation to resolve compared to when E runs on the initial partition P_i. An ER algorithm may run faster when resolving fewer (but larger) clusters. While not all ER algorithms that are $\mathcal{GI}$ will speed up from these two factors, we will see in Sect. 7 that the HC_B algorithm indeed benefits from Algorithm 2.

The complexity of Algorithm 2 can be computed by adding the cost for meeting partitions and the cost for running ER on clusters. In comparison with Algorithm 1, the additional cost is the outermost ER operation in Step 4. In practice, Algorithm 2 is slower than Algorithm 1, but can still be faster than running the ER algorithm E without rule evolution.

Proposition 3 *Algorithm 2 correctly returns an ER result $P_o \in \bar{E}(P_i, B_2)$.*

3.3 Materialization strategies

Until now, we have used a general strategy for rule materialization where we materialize on each conjunct. In this section, we list possible optimizations for materializations given more application-specific knowledge. Our list is by no means exhaustive, and the possible optimizations will depend on the ER algorithm and comparison rules.

A group of conjuncts is "stable" if they appear together in most comparison rules. As a result, the group can be materialized instead of all individual conjuncts. For example, if the conjuncts p_1, p_2, and p_3 are always compared as a conjunction in a person records comparison rule, then we can materialize on $p_1 \wedge p_2 \wedge p_3$ together rather than on the three conjuncts separately. Hence, the time and space overhead of materialization can be saved.

If we know the pattern of how the comparison rule will evolve, we can also avoid materializing on all conjuncts. In the ideal case where we know that the comparison rule can only get stricter, we do not have to save any additional materializations other than the ER result of the old comparison rule. Another scenario is when we are only changing the postfix of the old comparison rule, so we only need to materialize on all the prefixes of the old comparison rule. For example, if we have the comparison rule $p_1 \wedge p_2 \wedge p_3$, then we can materialize on p_1, $p_1 \wedge p_2$, and $p_1 \wedge p_2 \wedge p_3$. If the ER algorithm is both $\mathcal{RM}$ and $\mathcal{CF}$, then the ER result of $p_1 \wedge p_2$ can be computed efficiently from the ER result of p_1, and the ER result of $p_1 \wedge p_2 \wedge p_3$ from that of $p_1 \wedge p_2$.

4 Data evolution

Until now, we have studied how ER can be incremental on comparison rules. In this section, we focus on how ER can be incremental on newly incoming data. For example, if the ER result for the input $P_i = \{\{r_1\}, \{r_2\}\}$ is $P_o = \{\{r_1, r_2\}\}$ and we have a partition of new records $P'_i = \{\{r_3\}, \{r_4\}\}$ to resolve, we would like to quickly resolve the entire partition $\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}\}$ by exploiting the intermediate output P_o.

The general incremental ($\mathcal{GI}$) property (see Definition 6) can directly be used for incremental ER on new data. Suppose we use a match-based clustering ER algorithm E along with a Boolean comparison rule B. Given a previous ER result P_o (produced from P_i) and a new partition P'_i, the incremental data algorithm computes the ER result of $P_i \cup P'_i$ by computing $E(P_o \cup P'_i, B)$ (instead of $E(P_i \cup P'_i, B)$). In our example above, the ER result for $\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}\}$ can be computed with $E(P_o \cup P'_i, B) = E(\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}, B)$. We now prove the correctness of incrementally applying the ER algorithm on a previous ER result.

Proposition 4 *If the ER algorithm E is $\mathcal{GI}$, then given a previous ER result P_o (produced from P_i) and a new partition P'_i, the incremental data algorithm correctly produces an ER result of $P_i \cup P'_i$.*

While running ER from the intermediate result P_o has the same complexity as running ER from scratch, the time saved for producing P_o can be significant. Continuing our example above, suppose the ER algorithm E compares all pairs of clusters and merges the ones that match with each other. Also assume that, whenever E compares two clusters, it simply compares the records with the smallest IDs (e.g., a record r_2 has an ID of 2) from the two clusters using B. Since the subset $\{\{r_1\}, \{r_2\}\}$ has already been resolved into $P_o = \{\{r_1, r_2\}\}$, the incremental data algorithm can run E on the input $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$, which results in 3 record comparisons. On the other hand, if E started from scratch, it would have to resolve $\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}\}$ and thus perform 6 record comparisons.

However, the incremental data algorithm is not always faster than starting ER from scratch. Suppose that E still matches clusters the same way as above, but is now defined to re-resolve all the records of its input from scratch. Obviously, E still satisfies $\mathcal{GI}$. However, when E is given $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$ as input, it will start resolving $\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}\}$ (ignoring the intermediate result P_o) and perform all the 6 record comparisons. Hence, while the $\mathcal{GI}$ property is a pre-requisite for the correctness of incremental data evolution, it does not necessarily guarantee better efficiency.

5 Variations

In this section, we propose rule and data evolution techniques for two variations of the ER model. The first model clusters records based on their relative distances. The second model generates pairs of matching records instead of a partition of records. We will show that the properties and evolution algorithms in Sects. 3 and 4 carry over to these new scenarios.

5.1 Distance-based evolution

We consider rule and data evolution techniques on distance-based clustering where records are clustered based on their relative distances instead of the Boolean match results used in the match-based clustering model. We first define our comparison rule as a distance function. We then define the notion of strictness between distance comparison rules and define properties analogous to those in Sect. 2.2. Finally, we provide a model on how the distance comparison rule can evolve and present our rule evolution techniques.

5.1.1 Distance-based clustering model

In the distance-based clustering model, records are clustered based on their relative distances with each other. The comparison rule is now defined as a commutative distance function D that returns a non-negative distance between two records instead of a Boolean function as in Sect. 2.1. For example, the distance between two person records may be the sum of the distances between their names, addresses, and phone numbers. The details on how exactly D is used for the clustering differs for each ER algorithm. In hierarchical clustering using distances [21], the closest pairs of records are merged first until a certain criterion is met. A more sophisticated approach [6] may cluster a set of records that are closer to each other compared to records outside, regardless of the absolute distance values. Other than using a distance comparison rule instead of a Boolean comparison rule, the definition of a valid ER algorithm remains the same as Definition 1.

In order to support rule evolution, we model D to return a *range* of possible non-negative distances instead of a single non-negative distance. For example, the distance $D(r_1, r_2)$ can be all possible distances within the range [13, 15]. We denote the minimum possible value of $D(r_1, r_2)$ as $D(r_1, r_2).min$ (in our example, 13) and the maximum value as $D(r_1, r_2).max$ (in our example, 15). As a result, an ER algorithm that only supports single value distances must be extended to support ranges of values. The extension is specific to the given ER algorithm. (We provide concrete examples of extensions in Sect. 6.2.) However, in the case where the distance comparison rule only returns single value ranges, the extended algorithm must be identical to the original ER algorithm. Thus, the extension for general distances is only needed for rule evolution and does not change the behavior of the original ER algorithm.

A rule evolution occurs when a distance comparison rule D_1 is replaced by a new distance comparison rule D_2. We define the notion of relative strictness between distance comparison rules analogous to Definition 2.

Definition 7 A distance comparison rule D_1 is *stricter than* another rule D_2 (denoted as $D_1 \leq D_2$) if $\forall r, s, D_1(r, s).min \geq D_2(r, s).min$ and $D_1(r, s).max \leq D_2(r, s).max$.

That is, D_1 is stricter than D_2 if its distance range is always within that of D_2 for any record pair. For example, if $D_2(r, s)$ is defined as all the possible distance values within $[D_1(r, s).min-1, D_1(r, s).max+1]$, then $D_1 \leq D_2$ (assuming that $D_1(r, s).min \geq 1$).

5.1.2 Properties

We use properties analogous to $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{GI}$, and $\mathcal{OI}$ from Sect. 2.2 for the distance-based clustering model. The only differences are that we now use distance comparison rules instead of Boolean comparison rules (hence we must replace all B's with D's) and Definition 7 instead of Definition 2 for comparing the strictness between distance comparison rules. To show how the properties hold in practice, we define two distance-based clustering algorithms – HC_{DS} and HC_{DC}— in Sect. 6.2.

5.1.3 Rule evolution

While we used the CNF structures of comparison rules to perform rule evolution in Sect. 3.2, the distance comparison rules are not Boolean expressions. Instead, we define a model on how the distance comparison rule can evolve. We assume that each distance $D_1(r, s)$ changes by at most $f(D_1(r, s))$ where f is a positive function that can be provided by a domain expert who knows how much D_1 can change. Examples of f include a constant value (i.e., each distance can change by at most some constant c) or a certain ratio of the original distance (i.e., each distance can change by at most X percent). As a result, $D_1(r, s).max + f(D_1(r, s)) \geq D_2(r, s).max$ and $D_1(r, s).min - f(D_1(r, s)) \leq D_2(r, s).min$. As a practical example, suppose that D_1 returns the sum of the distances for the names, addresses, and zip codes, and D_2 returns the sum of the distances for the names, addresses, and phone numbers. If we restrict the zip code and phone number distances to be at most 10, then when D_1 evolves to D_2, we can set $f = 10$. Or if the zip code and phone number distances are always within 20% of the D_1 distance, then $f = 0.2 \times D_1$.

Given D_1 and D_2, we can now define a third distance comparison rule $D_3(r, s) = [max\{D_1(r, s).min - f(D_1(r, s)), 0\}, D_1(r, s).max + f(D_1(r, s))]$, which satisfies $D_3 \geq D_1$ and $D_3 \geq D_2$. (Notice that our definition ensures all the possible distances of D_3 to be non-negative.) Compared with the Boolean clustering model, rule D_3 acts as the "common conjuncts" between D_1 and D_2. As a result, we now materialize the ER result of D_3, $E(P_i, D_3)$, instead of the ER results for all the conjuncts in the first comparison rule. We also update Algorithm 1 in Sect. 3.2.1 by replacing Step 3 with "Partition $M \leftarrow H(D_3)$" where H is a hash table that only contains the result $E(P_i, D_3)$ for the comparison rule D_3.

Example We illustrate rule evolution for the HC_{DS} algorithm (see Sect. 6.2) using the updated Algorithm 1. Suppose we are given the input partition $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$ and the distance comparison rule D_1 where $D_1(r_1, r_2) = [2]$, $D_1(r_2, r_3) = [4]$, and $D_1(r_1, r_3) = [5]$. We use the threshold $T = 2$ for termination. If we are given $f(d) = 0.1 \times d$, D_3 is defined as $D_3(r_1, r_2) = [1.8, 2.2]$, $D_3(r_2, r_3) = [3.6, 4.4]$, and $D_3(r_1, r_3) = [4.5, 5.5]$. We then materialize the ER result $M = E(P_i, D_3)$. Among the records, only r_1 and r_2 match having $D_3(r_1, r_2).min = 1.8 \leq T = 2$. Once the clusters $\{r_1\}$ and $\{r_2\}$ merge, $\{r_1, r_2\}$ and $\{r_3\}$ do not match because $D_3(r_1, r_3).min = 4.5$ and $D_3(r_2, r_3).min = 3.6$, both exceeding T. Hence $M = \{\{r_1, r_2\}, \{r_3\}\}$. Suppose we are then given D_2 such that $D_2(r_1, r_2) = [2.2]$, $D_2(r_2, r_3) = [3.9]$, and $D_2(r_1, r_3) = [4.9]$ (notice that indeed $D_2 \leq D_3$). We then return $\bigcup_{c \in M} E(\{c' \in P_i | c' \subseteq c\}, D_2)$ using the same threshold $T = 2$. For the first cluster in M, we run $E(\{\{r_1\}, \{r_2\}\}, D_2)$. Since $D_2(r_1, r_2).min = 2.2 > T$, $\{r_1\}$ and $\{r_2\}$ do not merge. The next partition $\{\{r_3\}\}$ is a singleton, so our new ER result is $\{\{r_1\}, \{r_2\}, \{r_3\}\}$, which is identical to $E(P_i, D_2)$.

5.1.4 Data evolution

We can again exploit the $\mathcal{GI}$ property for incremental ER on new data. The data evolution techniques are the same as the ones in Sect. 4 except that we now use a distance comparison rule D instead of a boolean comparison rule B.

5.2 Pairs ER evolution

Until now, we have assumed that an ER algorithm clusters records and returns a partition. Many ER works [2,26,31], however, assume that ER returns the matching pairs of records instead of a clustering result. In this section, we thus consider an alternative ER model (called *pairs ER*) where an ER algorithm receives a set of records and returns a set of record pairs. Again, although we use a different ER model, we show in this section the ER properties and evolution techniques are analogous to those of clustering-based ER.

5.2.1 Pairs ER model

Given a set of records R and a Boolean comparison rule B, a pairs ER algorithm E returns the subset of pairs in the cross-product $R \times R$ excluding self-pairings that are matching pairs of records. We assume that E derives from a more general operator J_E that receives two sets of records S and T (along with B) and produces a set of pairs $J_E(S, T, B) \subseteq S \times T$. In the case where $S = T$, we exclude self-pairings. The pairs ER algorithm E thus resolves R by computing $J_E(R, R, B)$. Just like in Sect. 2.1, we assume that J_E is non-deterministic and may return more than one possible set of matching record pairs. Hence, we denote $\bar{J}_E(S, T, B)$ as the set of all possible $J_E(S, T, B)$ results. Finally, if E uses a distance function D for matching records, we can simply replace B with D.

Example We define a pairs ER algorithm (called *Join-Based*) that returns all the pairs of records that match according to B by performing pairwise comparisons. For instance, if $R = \{r_1, r_2, r_3\}$ and $B(r_1, r_2) = B(r_2, r_3) = \texttt{true}$ while $B(r_1, r_3) = \texttt{false}$, then $J_{Join}(R, R, B) = [(r_1, r_2), (r_2, r_3)]$. (We use square brackets to denote lists.)

5.2.2 Properties

Using the matching operator J_E of the pairs ER algorithm E, we can define the $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{GI}$, and $\mathcal{OI}$ properties for ER that are analogous to those in Sect. 2.2. (We omit the definitions for brevity.)

Proposition 5 *The Join-Based algorithm is $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{OI}$, and $\mathcal{GI}$.*

5.2.3 Rule evolution

Algorithm 3 performs rule evolution for pairs ER using the $\mathcal{RM}$ and $\mathcal{CF}$ properties and closely resembles Algorithm 1. Other rule evolution algorithms analogous to those for clustering ER algorithms can be defined in a similar fashion.

Revisiting our motivating example in Sect. 1, suppose that we are performing rule evolution for the *Join-Based* algorithm using Algorithm 3. We first materialize the two conjuncts of $B_1 = p_{name} \wedge p_{zip}$. The result of conjunct p_{name} is materialized as the list $[(r_1, r_2), (r_1, r_3), (r_2, r_3)]$ since the three records r_1, r_2, and r_3 match with each other. The con-

Algorithm 3 Rule evolution for pairs ER given $\mathcal{RM}$ and $\mathcal{CF}$

```
1: input: The comparison rules $B_1$, $B_2$, the pairs ER result for each
   conjunct of $B_1$, a hash table $H$ containing the pairs ER results for
   each conjunct in $B_1$
2: output: Pairs ER result using $B_2$
3: Pairs $I \leftarrow \bigcap_{conj \in Conj(B_1) \cap Conj(B_2)} H(conj)$
4: return $\bigcup_{c \in TransClosure(I)} J_E(c, c, B_2)$
```

junct p_{zip} (which also has exactly one predicate) is materialized as $[(r_1, r_2)]$ because only r_1 and r_2 have the same zip code. During the rule evolution using $B_2 = p_{name} \wedge p_{phone}$, we set $I = [(r_1, r_2), (r_1, r_3), (r_2, r_3)]$ and compute the transitive closure of I, which is $\{\{r_1, r_2, r_3\}\}$. For the only cluster $c = \{r_1, r_2, r_3\}$ in the transitive closure, we compute $J_E(c, c, B_2)$ and return $[(r_2, r_3)]$, which is the exact same solution as resolving all four records r_1, r_2, r_3, and r_4 using B_2.

Proposition 6 *Algorithm 3 correctly returns the set of record pairs* $J_o \in \bar{J}_E(R, R, B_2)$.

5.2.4 Data evolution

Just like in Sect. 4, we can exploit the $\mathcal{GI}$ property for incremental ER on new data. Suppose we use a pairs ER algorithm E along with a Boolean comparison rule B. Given a previous result $J_o = J_E(R, R, B)$ and a new set of records S, the incremental data algorithm performs this step: $J_o \cup J_E(R, S, B) \cup J_E(S, S, B)$.

Proposition 7 *If a pairs ER algorithm E is $\mathcal{GI}$, then given a previous ER result J_o (produced from R) and a new set of records S, the incremental data algorithm correctly produces an ER result of $R \cup S$.*

6 ER algorithms and their properties

We now define the match-based and distance-based ER algorithms in the venn diagram of Fig. 2 and prove why each ER algorithm satisfies its properties. All the proofs for the results in this section can be found in Appendix 10. As a reminder, we abbreviate the rule monotonic property as $\mathcal{RM}$, context free as $\mathcal{CF}$, general incremental as $\mathcal{GI}$, and order independent as $\mathcal{OI}$.

6.1 Match-based clustering

The match-based ER algorithms in Fig. 2 are SN, SN^2, HC_B, HC_B^2, HC_{BR}, and ME. While the original definitions of these ER algorithms assume a set of records S as an input, we provide simple extensions for the algorithms to accept a set of clusters P_i as in Definition 1. We then show which properties each ER algorithm satisfies. We note that the definitions of SN^2 and HC_B^2 can be found in Appendix 10.

6.1.1 ER algorithms

SN The sorted neighborhood (SN) algorithm [16] first sorts the records in P_i (i.e., we extract all the records from the clusters in P_i) using a certain key assuming that closer records in the sorted list are more likely to match. For example, suppose that we have the input partition $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$ and sort the clusters by their names (which are not visible in this example) in alphabetical order to obtain the list $[r_1, r_2, r_3]$. The SN algorithm then slides a fixed-sized window on the sorted list of records and compares all the pairs of clusters that are inside the same window at any point. If the window size is 2 in our example, then we compare r_1 with r_2 and then r_2 with r_3, but not r_1 with r_3 because they are never in the same window. We thus produce pairs of records that match with each other. We can repeat this process using different keys (e.g., we could also sort the person records by their address values). After collecting all the pairs of records that match, we perform a transitive closure on all the matching pairs of records to produce a partition P_o of records. For example, if r_1 matches with r_2 and r_2 matches with r_3, then we merge r_1, r_2, r_3 together into the output $P_o = \{\{r_1, r_2, r_3\}\}$.

Proposition 8 *The SN algorithm is $\mathcal{RM}$, but not $\mathcal{CF}$.*

HC$_\mathbf{B}$ Hierarchical clustering based on a Boolean comparison rule [3] (which we call HC_B) combines matching pairs of clusters in any order until no clusters match with each other. The comparison of two clusters can be done using an arbitrary function that receives two clusters and returns `true` or `false`, using the Boolean comparison rule B to compare pairs of records. For example, suppose we have the input partition $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$ and the comparison rule B where $B(r_1, r_2) = \texttt{true}$, $B(r_2, r_3) = \texttt{true}$, but $B(r_1, r_3) = \texttt{false}$. Also assume that, whenever we compare two clusters of records, we simply compare the records with the smallest IDs (e.g., a record r_2 has an ID of 2) from each cluster using B. For instance, when comparing $\{r_1, r_2\}$ with $\{r_3\}$, we return the result of $B(r_1, r_3)$. Depending on the order of clusters compared, the HC_B algorithm can merge $\{r_1\}$ and $\{r_2\}$ first, or $\{r_2\}$ and $\{r_3\}$ first. In the first case, the final ER result is $\{\{r_1, r_2\}, \{r_3\}\}$ (because the clusters $\{r_1, r_2\}$ and $\{r_3\}$ do not match), while in the second case, the ER result is $\{\{r_1\}, \{r_2, r_3\}\}$ (the clusters $\{r_1\}$ and $\{r_2, r_3\}$ do not match). Hence, $\bar{E}(P_i, B) = \{\{\{r_1, r_2\}, \{r_3\}\}, \{\{r_1\}, \{r_2, r_3\}\}\}$.

Proposition 9 *The HC_B algorithm is $\mathcal{CF}$, but not $\mathcal{RM}$.*

HC$_\mathbf{BR}$ We define the HC_{BR} algorithm as a hierarchical clustering algorithm based on a Boolean comparison rule (just like HC_B). In addition, we require the comparison rule to match two clusters whenever at least one of the records from the two clusters match according to B. (This property is equivalent to the representativity property in reference [3].) For example, a cluster comparison function that compares all the records between two clusters using B for an existential match is representative. That is, given two clusters $\{r_1, r_2\}$ and $\{r_3, r_4\}$, the cluster comparison function returns `true` if at least one of $B(r_1, r_3)$, $B(r_1, r_4)$, $B(r_2, r_3)$, or $B(r_2, r_4)$ returns `true`.

We can prove that the HC_{BR} is both $\mathcal{RM}$ and $\mathcal{CF}$ (see Proposition 11). We first define the notation of connectedness. Two records r and s are connected under B and P_i if there exists a sequence of records $[r_1(=r), \ldots, r_n(=s)]$ where for each pair (r_i, r_{i+1}) in the path, either $B(r_i, r_{i+1}) =$ `true` or $\exists c \in P_i$ s.t. $r_i \in c, r_{i+1} \in c$. Notice that connectedness is "transitive," i.e., if r and s are connected and s and t are connected, then r and t are also connected.

Lemma 5 *Two records r and s are connected under B and P_i if and only if r and s are in the same cluster in $P_o \in \bar{E}(P_i, B)$ using the HC_{BR} algorithm.*

We next prove that HC_{BR} returns a unique solution.

Proposition 10 *The HC_{BR} algorithm always returns a unique solution.*

Finally, we prove that HC_{BR} is both $\mathcal{RM}$ and $\mathcal{CF}$.

Proposition 11 *The HC_{BR} algorithm is both $\mathcal{RM}$ and $\mathcal{CF}$.*

ME The Monge–Elkan (ME) clustering algorithm (we define a variant of the algorithm in [24] for simplicity) first sorts the records in P_i (i.e., we extract all the records from the clusters in P_i) by some key and then starts to scan each record. For example, suppose that we are given the input partition $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$, and we sort the records in P_i by their names (which are not visible in this example) in alphabetical order into the sorted list of records $[r_1, r_2, r_3]$. Suppose we are also given the Boolean comparison rule B where $B(r_1, r_2) =$ `true`, but $B(r_1, r_3) =$ `false` and $B(r_2, r_3) =$ `false`. Each scanned record is then compared with clusters in a fixed-length queue. A record r matches with a cluster c if $B(r, s) =$ `true` for any $s \in c$. If the new record matches one of the clusters, the record and cluster merge, and the new cluster is promoted to the head of the queue. Otherwise, the new record forms a new singleton cluster and is pushed into the head of the queue. If the queue is full, the last cluster in the queue is dropped. In our example, if the queue size is 1, then we first add r_1 into the head of the queue, and then compare r_2 with $\{r_1\}$. Since r_2 matches with $\{r_1\}$, we merge r_2 into $\{r_1\}$. We now compare r_3 with the cluster $\{r_1, r_2\}$ in the queue. Since r_3 does not match with $\{r_1, r_2\}$, then we insert $\{r_3\}$ into the head of the queue and thus remove $\{r_1, r_2\}$. Hence, the only possible ER result is $\{\{r_1, r_2\}, \{r_3\}\}$ and thus $\bar{E}(P_i, B) = \{\{\{r_1, r_2\}, \{r_3\}\}\}$. In general, ME always returns a unique partition.

Proposition 12 *The ME algorithm does not satisfy $\mathcal{RM}$ or $\mathcal{CF}$.*

6.1.2 More properties

We continue the construction of the venn diagram in Fig. 2. We first add the $\mathcal{OI}$ property into the venn diagram. Proposition 13 shows that the $\mathcal{OI}$ property includes the $\mathcal{RM}$ property. Proposition 14 shows that the ME algorithm is $\mathcal{OI}$, but not $\mathcal{RM}$. Also, $\mathcal{OI}$ partially overlaps with $\mathcal{CF}$, but does not contain it. According to Proposition 15, the HC_{BR} algorithm is both $\mathcal{OI}$ and $\mathcal{CF}$. According to Proposition 16, the HC_B algorithm is $\mathcal{CF}$, but not $\mathcal{OI}$. Finally, Proposition 14 shows that the ME algorithm is $\mathcal{OI}$, but not $\mathcal{CF}$.

Proposition 13 *Any ER algorithm that is $\mathcal{RM}$ is also $\mathcal{OI}$.*

Proposition 14 *The ME algorithm is $\mathcal{OI}$ and $\mathcal{GI}$, but not $\mathcal{RM}$ or $\mathcal{CF}$.*

Proposition 15 *The HC_{BR} algorithm is $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{GI}$, and $\mathcal{OI}$.*

Proposition 16 *The HC_B algorithm is $\mathcal{CF}$ and $\mathcal{GI}$, but not $\mathcal{OI}$ (and consequently not $\mathcal{RM}$).*

We now add the $\mathcal{GI}$ property into the venn diagram in Fig. 2. The $\mathcal{GI}$ property partially intersects with the other three properties $\mathcal{RM}$, $\mathcal{CF}$, and $\mathcal{OI}$ and does not include any of them. Proposition 15 shows that HC_{BR} is $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{GI}$, and $\mathcal{OI}$. Hence, $\mathcal{GI}$ intersects with the other three properties. Proposition 16 shows that the HC_B algorithm is $\mathcal{GI}$, but not $\mathcal{OI}$ (and consequently not $\mathcal{RM}$). Proposition 14 shows that the ME algorithm is $\mathcal{GI}$, but not $\mathcal{CF}$. Hence, none of the other three properties include $\mathcal{GI}$. Proposition 17 shows the existence of an ER algorithm that is $\mathcal{RM}$ (the same algorithm is also $\mathcal{OI}$), but not $\mathcal{GI}$. Proposition 18 shows the existence of an ER algorithm that is $\mathcal{CF}$, but not $\mathcal{GI}$. Hence, $\mathcal{GI}$ does not include any of the other three properties.

Proposition 17 *There exists an ER algorithm that is $\mathcal{RM}$ (and consequently $\mathcal{OI}$ as well), but not $\mathcal{GI}$ or $\mathcal{CF}$.*

Proposition 18 *There exists an ER algorithm that is $\mathcal{CF}$ and $\mathcal{OI}$, but not $\mathcal{GI}$ or $\mathcal{RM}$.*

Finally, Proposition 19 shows that the SN algorithm is $\mathcal{RM}$, $\mathcal{OI}$, and $\mathcal{GI}$, but not $\mathcal{CF}$.

Proposition 19 *The SN algorithm is $\mathcal{RM}$ (and consequently $\mathcal{OI}$) and $\mathcal{GI}$, but not $\mathcal{CF}$.*

6.2 Distance-based clustering

We now define the two distance-based ER algorithms in Fig. 2 and show which properties they satisfy.

HC_{DS} The single-link hierarchical clustering algorithm [12,21] (HC_{DS}) merges the closest pair of clusters (i.e., the two clusters that have the smallest distance) into a single cluster until the smallest distance among all pairs of clusters exceeds a certain threshold T. When measuring the distance between two clusters, the algorithm takes the smallest possible distance between records within the two clusters. Suppose we have the input partition $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$ where $D(r_1, r_2) = 2$, $D(r_2, r_3) = 4$, and $D(r_1, r_3) = 5$ (we later extend HC_{DS} to support ranges of distances) with $T = 2$. The HC_{DS} algorithm first merges r_1 and r_2, which are the closest records and have a distance smaller or equal to T, into $\{r_1, r_2\}$. The cluster distance between $\{r_1, r_2\}$ and $\{r_3\}$ is the minimum of $D(r_1, r_3)$ and $D(r_2, r_3)$, which is 4. Since the distance exceeds T, $\{r_1, r_2\}$ and $\{r_3\}$ do not merge, the final result is $\{\{r_1, r_2\}, \{r_3\}\}$.

We extend the HC_{DS} algorithm by allowing ranges of distances to be returned by a distance comparison rule, but only comparing the minimum value of a range with either another range or the threshold T. That is, $D(r, s)$ is considered a smaller distance than $D(u, v)$ if $D(r, s).min \leq D(u, v).min$. Also, $D(r, s)$ is considered smaller than T if $D(r, s).min \leq T$. For example, [3, 5] < [4, 4] because 3 is smaller than 4, and $[3, 5] > T = 2$ because 3 is larger than 2. The extended HC_{DS} algorithm is trivially identical to the original HC_{DS} algorithm when D only returns a single value.

Proposition 20 shows that the HC_{DS} algorithm is $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{GI}$, and $\mathcal{OI}$. As a result, the HC_{DS} algorithm can use Algorithm 1 (with minor changes; see Sect. 5.1.3) for rule evolution.

Proposition 20 *The extended HC_{DS} algorithm is $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{GI}$, and $\mathcal{OI}$.*

HC_{DC} The complete-link hierarchical clustering algorithm [21] (HC_{DC}) is identical to the HC_{DS} algorithm except in how it measures the distance between two clusters. While the HC_{DS} algorithm chooses the smallest possible distance between records within the two clusters, the HC_{DC} algorithm takes the largest possible distance instead. For example, the cluster distance between $\{r_1, r_2\}$ and $\{r_3\}$ is the maximum of $D(r_1, r_3)$ and $D(r_2, r_3)$. We use the same extension used in HC_{DS} to support ranges of values for distances where only the minimum values of each range are compared to other ranges or thresholds.

Proposition 21 shows that the extended HC_{DC} algorithm is $\mathcal{CF}$ and $\mathcal{OI}$, but not $\mathcal{RM}$ or $\mathcal{GI}$. As a result, the extended HC_{DC} algorithm cannot use Algorithms 1 or 2 for rule evolution.

Proposition 21 *The extended HC_{DC} algorithm is $\mathcal{CF}$, but not $\mathcal{RM}$, $\mathcal{OI}$, or $\mathcal{GI}$.*

7 Experimental evaluation

We evaluate our rule and data evolution techniques for ER based on clustering. Since data incremental ER algorithms have been studied and evaluated in our previous work [3], our focus in this section is on rule evolution. However, in Sect. 7.8, we briefly illustrate the potential gains of incremental data processing. We do not present experimental results for pairs ER evolution since they are analogous to those of clustering-based ER.

Evaluating rule evolution is challenging since the results depend on many factors including the ER algorithm, the comparison rules, and the materialization strategy. Obviously, there are many cases where evolution and/or materialization are not effective, so our goal in this section is to show there are realistic cases where they can pay off, and that in some cases the savings over a naïve approach can be significant. (Of course, as the saying goes, “your mileage may vary”!) The savings can be very important in scenarios where datasets are large and where it is important to obtain a new ER result as quickly as possible (think of national security applications where it is critical to respond to new threats as quickly as possible).

For our evaluation, we assume that *blocking* [25] is used, as it is in most ER applications with massive data. With blocking, the input records are divided into separate blocks using one or more key fields. For instance, if we are resolving products, we can partition them by category (books, movies, electronics, etc). Then, the records within one block are resolved independently from the other blocks. This approach lowers accuracy because records in separate blocks are not compared, but makes resolution feasible. (See [22,30] for more sophisticated approaches). From our point of view, the use of blocking means that we can read a full block (which can still span many disk blocks) into memory, perform resolution (naïve or evolutionary), and then move on to the next block. In our experiments, we thus evaluate the cost of resolving a single block. Keep in mind that these costs should be multiplied by the number of blocks.

There are three metrics that we use to compare ER strategies: CPU, IO, and storage costs. (Except for Sect. 7.7, we do not consider accuracy since our evolution techniques do not change the ER result, only the cost of obtaining it.) We discuss CPU and storage costs in the rest of this section, leaving a discussion of IO costs to Sect. 7.2. In general, CPU costs tend to be the most critical due to the quadratic nature of the

ER problem, and because matching/distance rules tend to be expensive. In Sect. 7.2, we argue that IO costs do not vary significantly with or without evolution and/or materialization, further justifying our focus here on CPU costs.

We start by describing our experimental setting in Sect. 7.1. Then, in Sects. 7.3 and 7.4, we discuss the CPU costs of ER evolution compared to a naïve approach (ignoring materialization costs, if any). In Sect. 7.5, we consider the CPU and space overhead of materializing partitions. Note that we do not discuss the orthogonal problem of *when* to materialize (a problem analogous to selecting what views to materialize). In Sect. 7.6, we discuss total costs, including materialization and evolution. In Sect. 7.7, we consider scenarios where the necessary properties do not hold. Finally, in Sect. 7.8, we evaluate data evolution.

7.1 Experimental setting

We experiment on a comparison shopping dataset provided by Yahoo! Shopping and a hotel dataset provided by Yahoo! Travel. We evaluated the following ER algorithms: SN, HC_B, HC_{BR}, ME, HC_{DS}, and HC_{DC}. Our algorithms were implemented in Java, and our experiments were run on a 2.4GHz Intel(R) Core 2 processor with 4GB of RAM.

Real Data The comparison shopping dataset we use was provided by Yahoo! Shopping and contains millions of records that arrive on a regular basis from different online stores and must be resolved before they are used to answer customer queries. Each record contains various attributes including the title, price, and category of an item. We experimented on a random subset of 3,000 shopping records that had the string “iPod” in their titles and a random subset of 1 million shopping records. We also experimented on a hotel dataset provided by Yahoo! Travel where tens of thousands of records arrive from different travel sources (e.g., Orbitz.com) and must be resolved before they are shown to the users. We experimented on a random subset of 3,000 hotel records located in the United States. While the 3K shopping and hotel datasets fit in memory, the 1 million shopping dataset did not fit in memory and had to be stored on disk.

Comparison Rules Table 3 summarizes the comparison rules used in our experiments. The *Type* column indicates whether the comparison rules are Boolean comparison rules or distance comparison rules. The *Data* column indicates the data source: shopping or hotel data. The *Comparison rules* column indicates the comparison rules used. The first two rows define the Boolean comparison rules used on the shopping and hotel datasets. For the shopping datasets, B_1^S compares the titles and categories of two shopping records while B_2^S compares the titles and prices of shopping records. For the hotel data, B_1^H compares the states, cities, zip codes, and names of two hotel records. The B_2^H rule compares the states, cities, zip codes, and street addresses of two hotel records. The last two rows define the distance comparison rules for the two datasets. For the shopping data, D_1^S measures the Jaro distance [31] between the titles of two shopping records while D_2^S randomly alters the distance of D_1^S by a maximum ratio of 5%. The Jaro distance returns a value within the range [0, 1] and gives higher values for closer records. For the hotel data, D_1^H sums the Jaro distance between the names of two records and the equality distance between the cities of two records weighted by 0.05. We define the equality distance to return 1 if two values are exactly the same and 0 if they are not the same. The D_2^H rule sums the Jaro distance between names with the equality distance between the zip codes of two records weighted by 0.05. As a result, the D_1^H distance can alter by at most the constant 0.05.

Table 3 Comparison Rules

Type	Data	Comparison rules
Boolean	Shopping	$B_1^S : p_{ti} \wedge p_{ca}$
		$B_2^S : p_{ti} \wedge p_{pr}$
Boolean	Hotel	$B_1^H : p_{st} \wedge p_{ci} \wedge p_{zi} \wedge p_{na}$
		$B_2^H : p_{st} \wedge p_{ci} \wedge p_{zi} \wedge p_{sa}$
Distance	Shopping	$D_1^S : Jaro_{ti}$
		$D_2^S : Jaro_{ti}$ changes randomly within 5%
Distance	Hotel	$D_1^H : Jaro_{na} + 0.05 \times Equals_{ci}$
		$D_2^H : Jaro_{na} + 0.05 \times Equals_{zi}$

Table 4 ER and rule evolution algorithms tested

ER algorithm	Section	Rule evolution algorithm used
SN	6.1.1	Alg. for SN in Sect. 3.2.2
HC_B	6.1.1	Alg. 2
HC_{BR}	6.1.1	Alg. 1
ME	6.1.1	Alg. 1
HC_{DS}	6.2	Alg. 1 (for distance-based clustering)
HC_{DC}	6.2	Alg. 1 (for distance-based clustering)

ER and Rule Evolution Algorithms We experiment rule evolution on the following ER algorithms: SN, HC_B, HC_{BR}, ME, HC_{DS}, and HC_{DC}. Table 4 summarizes for each ER algorithm which section it was defined in and which rule evolution algorithm is used. The HC_{DS} and HC_{DC} distanced-based clustering algorithms terminate when the minimum distance between clusters is smaller than the threshold 0.95 (recall that *closer* records have *higher* Jaro + equality distances). Although the ME and HC_{DC} algorithms do not satisfy the $\mathcal{RM}$ property, we can still use Algorithm 1 to efficiently produce new ER results with small loss in accuracy. Notice that, although ME is $\mathcal{GI}$, Algorithm 2 is not efficient because of the way ME extracts all records from the input partition P_i (without exploiting any of the clusters in P_i) and sorts them again. Both the HC_{DS} and HC_{DC} algorithms use Algorithm 1 adjusted for the distance-based clustering model (see Sect. 5.1.3).

7.2 Evaluating IO costs

We discuss the corresponding IO costs and argue that the materialization IO costs are less significant than the CPU costs. Using our blocking framework, we can analyze the overall runtime of an ER process. The basic operations of an ER process are described in Table 5. The operations are categorized depending on whether they are disk IO consuming operations or CPU time-consuming operations.

To compare the overall performance of an ER process using rule evolution and a naïve ER process without rule evolution, we consider the scenario where we run ER once using an old comparison rule and then perform one rule evolution using a new comparison rule. A naïve ER process without rule evolution would roughly require initializing the records, creating the blocks, and reading and resolving the blocks twice. An ER process using rule evolution on the other hand would require the same process above plus the additional work of creating and using rule materializations minus running ER on all blocks during the rule evolution. The decompositions of the two approaches for our one rule evolution scenario are shown in Table 6. Notice that the listed operations are not necessarily run sequentially. For example, for the naïve approach, the R_B and E operations are actually interleaved because each block is read and then resolved before the next block is read.

Table 5 Basic operations in blocking ER framework

Oper.	Description
IO time-consuming operations	
R_F	Read records from input file
R_B	Read all blocks to memory
W_B	Write out all blocks to disk
R_M	Read all materializations to memory
W_M	Write all materializations to disk
O	Write the output ER result to disk
CPU time-consuming operations	
I	Initialize records (trim attributes not used in rules)
E	Run ER on all blocks (one block at a time)
M	Create materializations for all blocks (one at a time)
V	Run rule evolution (using materializations) on all blocks (one at a time)

Table 6 Decomposition of ER processes for one rule evolution

ER process	Decomposition
Naïve	$R_F, I, W_B, R_B, E, O, R_B, E, O$
Using rule evolution	$R_F, I, W_B, R_B, E, O, M, W_M, R_B, R_M, V, O$

The IO overhead of using rule evolution compared with the IO cost of the naïve approach can thus be written as $\frac{R_M+W_M}{R_F+W_B+2\times R_B+2\times O}$. Since the size of the materializations is usually much smaller than the size of the entire set of records (see Sect. 3.1), the additional IOs for rule evolution are also smaller than the IOs for reading and writing the blocks. Thus, the IO costs do not vary significantly with or without evolution and/or materialization.

7.3 Rule evolution efficiency

We first focus on the CPU time cost of rule evolution (exclusive of materialization costs, if any) using blocks of data that fit in memory. For each ER algorithm, we use the best evaluation scheme (see Sect. 7.1) given the properties of the ER algorithm. Table 7 shows the results. We run the ER algorithms SN, HC_B, and HC_{BR} using the Boolean comparison rules in Table 3 on the shopping and hotel datasets. When evaluating each comparison rule, the conjuncts involving string comparisons (i.e., p_{ti}, p_{na}, and p_{sa}) are evaluated last because they are more expensive than the rest of the conjuncts. We also run the HC_{DS} algorithm using the distance comparison rules in Table 3 on the two datasets. Each column head in Table 7 encodes the dataset used and the number of records resolved in the block. For example, Sh1K means 1,000 shopping records while Ho3K means 3,000 hotel records. The top five rows of data show the runtime results of the naïve approach while the bottom five rows show the runtime improvements of rule evolution compared with the naïve approach. Each runtime improvement is computed by dividing the naïve approach runtime by the rule evolution runtime. For example, the HC_{BR} algorithm takes 3.56 seconds to run on 1K shopping records and rule evolution is 162 times faster (i.e., having a runtime of $\frac{3.56}{162} = 0.022$ seconds).

As one can see in Table 7, the improvements vary widely but in many cases can be very significant. For the shopping dataset, the HC_{BR}, and HC_{DS} algorithms show up to orders of magnitude of runtime improvements. The SN algorithm has a smaller speedup because SN itself runs efficiently. The HC_B algorithm has the least speedup (although still a speedup). While the rule evolution algorithms for SN, HC_{BR}, and HC_{DS} only need to resolve few clusters at a time (i.e., each $\{c' \in P_i | c' \subseteq c\}$ in Algorithm 1), Algorithm 2 for the HC_B algorithm also needs to run an outermost ER operation (Step 4) to resolve the clusters produced by the inner ER operations. The hotel data results show worse run-

Table 7 ER algorithm and rule evolution runtimes

ER algorithm	Sh1K	Sh2K	Sh3K	Ho1K	Ho2K	Ho3K
ER algorithm runtime (seconds)						
SN	0.094	0.152	0.249	0.012	0.027	0.042
HC_B	1.85	7.59	17.43	0.386	2.317	5.933
HC_{BR}	3.56	19.37	48.72	0.322	1.632	4.264
HC_{DS}	8.33	40.38	111	5.482	27.96	73.59
Rule evolution runtime (seconds)						
SN	0.023	0.036	0.056	0.01	0.014	0.021
HC_B	1.23	4.125	8.42	0.304	1.782	4.672
HC_{BR}	0.022	0.024	0.04	0.009	0.012	0.018
HC_{DS}	0.028	0.057	0.121	0.017	0.056	0.135
Ratio of ER algorithm runtime to rule evolution runtime						
SN	4.09	4.22	4.45	1.2	1.93	2
HC_B	1.5	1.84	2.07	1.27	1.3	1.27
HC_{BR}	162	807	1218	36	136	237
HC_{DS}	298	708	918	322	499	545

time improvements overall because the ER algorithms without rule evolution ran efficiently.

7.4 Common rule strictness

The key factor of the runtime savings in Sect. 7.3 is the strictness of the "common comparison rule" between the old and new comparison rules. For match-based clustering, the common comparison rule between B_1 and B_2 comprises the common conjuncts $Conj(B_1) \cap Conj(B_2)$. For distance-based clustering, the common comparison rule between D_1 and D_2 is D_3, as defined in Sect. 5.1.3. A stricter rule is more selective (fewer records match or fewer records are within the threshold) and leads to smaller clusters in a resolved result. If the common comparison rule yields smaller clusters, then in many cases, the resolution that starts from there will have less work to do.

By changing the thresholds used by the various predicates, we can experiment with different common rule strictness, and Fig. 3 summarizes some of our findings. The horizontal axis shows the strictness of the common rule: It gives the ratio of record pairs placed by the common rule within in a cluster to the total number of record pairs. For example, if an ER algorithm uses p_{ti} to produce 10 clusters of size 10, then the strictness is $\frac{10\times\binom{10}{2}}{\binom{100}{2}} = 0.09$. The lower the ratio is, the stricter the common rule, and presumably, fewer records need to be resolved using the new comparison rule.

The vertical axis in Fig. 3 shows the runtime improvement (vs. naïve), for four algorithms using our shopping data comparison rules in Table 3. The runtime improvement is

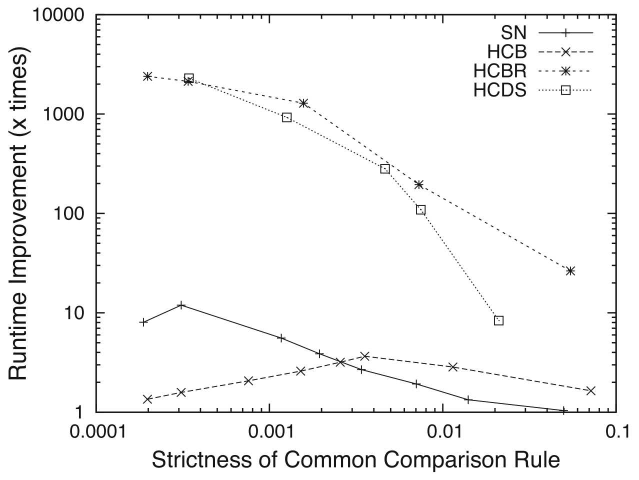

Fig. 3 Degree of change impact on runtime, 3K shopping records

computed as the runtime of the naïve approach computing the new ER result divided by the runtime of rule evolution. As expected, Algorithms SN, HC_{BR}, and HC_{DS} achieve significantly higher runtime improvements as the common comparison rule becomes stricter. However, the HC_B algorithm shows a counterintuitive trend (performance decreases as strictness increases). In this case, there are two competing factors. On one hand, having a stricter common comparison rule improves runtime for rule evolution because the computation of each $E(\{c' \in P_i | c' \subseteq c\}, B_2)$ in Step 4 becomes more efficient. On the other hand, a common comparison rule that is too strict produces many clusters to resolve for the outermost ER operation in Step 4, increasing the overall runtime. Hence, the increasing line eventually starts decreasing as strictness decreases.

7.5 Materialization overhead

In this section, we examine the CPU and space overhead of materializations, independent of the question of what conjuncts should be materialized. Recall that materializations are done as we perform the initial resolution on records S. Thus, the materialization can piggyback on the ER work that needs to be done anyway. For example, the parsing and initialization of records can be done once for the entire process of creating all materializations and running ER for the old comparison rule. In addition, there are other ways to amortize work, as the resolution is concurrently done for the old rule and the conjuncts we want to materialize. For example, when materializing for the SN and ME algorithms, the sorting of records is only done once. For the HC_B and HC_{BR} algorithms, we cache the merge information of records. For the HC_{DS} and HC_{DC} algorithms, the pairwise distances between records are only computed once. We can also compress the storage space needed by materializations by storing partitions of record IDs.

Table 8 shows the time and space overhead of materialization in several representative scenarios. In particular, we use Algorithms SN, HC_B, HC_{BR}, and HC_{DS} on 3K shopping and hotel records and assume *all* conjuncts in the old rule are materialized.

The *Time O/H* columns show the time overhead where each number is produced by dividing the materialization CPU time by the CPU runtime for producing the old ER result. For example, materialization time for the SN algorithm on 3K shopping records is 0.52x the time for running $E(P_i, B_1^S)$ using SN. Hence, the total time to compute $E(P_i, B_1^S)$ and materialize all the conjuncts of B_1^S is $1+0.52 = 1.52$ times the runtime for $E(P_i, B_1^S)$ only. The numbers in parentheses show the time overhead when we do *not* materialize the most expensive conjunct. That is, for SN, HC_B, and HC_{BR} in the shopping column, we only materialize p_{ca}; in the hotel column, we only materialize p_{st}, p_{ci}, and p_{zi} (without p_{na}).

For the shopping dataset, the SN and HC_B algorithms have time overheads less than 2 (i.e., the number of conjuncts in B_1^S) due to amortization. For the same reason, HC_{DS} has a time overhead below 1. The HC_{BR} algorithm has a large overhead of 11x because each common conjunct tends to produce larger clusters compared with $E(P_i, B_1^H)$, and HC_{BR} ran slowly when larger clusters were compared using the expensive p_{ti} conjunct.

The hotel dataset shows similar time overhead results, except that the time overheads usually do not exceed 4 (i.e., the number of conjuncts in B_1^H) for the match-based clustering algorithms.

The *Space O/H* columns show the space overhead of materialization where each number was produced by dividing the memory space needed for storing the materialization by the memory space needed for storing the old ER result. For example, the materialization space for the SN algorithm on 3K shopping records is 0.28x the memory space taken by $E(P_i, B_1^S)$ using SN. The total required space is thus 1+0.28 = 1.28 times the memory space needed for $E(P_i, B_1^S)$. The space overhead of materialization is small in general because we only store records by their IDs.

Table 8 Time overhead (ratio to old ER algorithm runtime) and space overhead (ratio to old ER result) of rule materialization, 3K records

ER alg.	Sh3K Time O/H	Space O/H Time O/H	Ho3K	Space O/H
SN	0.52 (0.02)	0.28	1.14 (0.27)	0.14
HC_B	0.87 (0.04)	0.14	3.18 (0.71)	0.1
HC_{BR}	11 (3E-6)	0.14	13.28 (1.06)	0.1
HC_{DS}	0.44	0.07	0.61	0.02

7.6 Total runtime

The speedups achievable at evolution time must be balanced against the cost of materializations during earlier resolutions. The materialization cost of course depends on what is materialized: If we do not materialize any conjuncts, as in our initial example in Sect. 1, then clearly there is no overhead. At the other extreme, if the initial rule B_1 has many conjuncts and we materialize all of them, the materialization cost will be higher. If we have application knowledge and know what conjuncts are “stable” and likely to be used in future rules, then we can only materialize those. Then, there is also the amortization factor: If a materialization can be used many times (e.g., if we want to explore many new rules that share the materialized conjunct), then the materialization cost, even if high, can be amortized over all the future resolutions.

We study the total run time (CPU and IO time for original resolution plus materializations plus evolution) for several scenarios. We experiment on 0.25 to 1 million shopping records (multiple blocks are processed). Our results illustrate scenarios where materialization does pay off. That is, materialization and evolution lowers the total time, as compared to the naïve approach that runs ER from scratch each time. Of course, one can also construct scenarios where materialization does not pay off.

We measure the total runtimes of ER processes as defined in Sect. 7.2 where we run ER once using an old comparison rule and then perform one rule evolution using a new comparison rule. We experimented on 0.25 to 1 million random shopping records and used the following Boolean comparison rules for the SN, HC_B, and HC_{BR} algorithms: $B_1 = p_{ca} \wedge p_{ti}$ (same as B_1^S in Table 3) and $B_2 = p_{ca} \wedge p_{pr}$. In

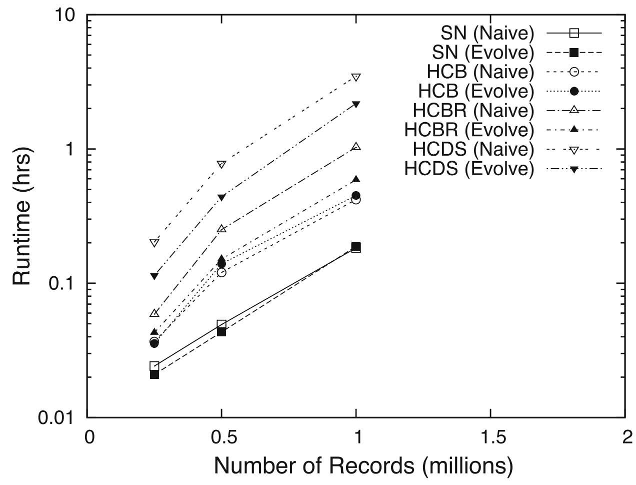

Fig. 4 Scalability, 1M shopping records

addition, we only materialized on the conjunct p_{ca} instead of on both conjuncts in B_1. The time overheads for materializing p_{ca} were shown in parentheses in Fig. 4. For the HC_{DS} algorithm, we used D_1^S and D_2^S in Table 3. We used minhash signatures [17] for distributing the records into blocks. For the shopping dataset, we extracted 3-grams from the titles of records. We then generated a minhash signature for each records, which is an array of integers where each integer is generated by applying a random hash function to the 3-gram set of the record.

Figure 4 shows our total time results where we measured the total runtimes of running ER on B_1 and then evolving once to B_2. Each rule evolution technique and its corresponding naïve approach use the same shape for points in their plots. For example, the rule evolution runtime plot for the SN algorithm uses *white square* points while the naïve SN approach uses *black square* points. In addition, all the naïve approach plots use white shapes while the rule evolution plots use black shapes. Our results show that the total runtimes for the SN and HC_B algorithms do not change much because the runtime benefits of using rule evolution more or less cancels out the runtime overheads of using rule evolution. For the HC_{BR} and HC_{DS} algorithms, however, the runtime benefits of rule evolution clearly exceed the overheads. While we have shown the worst case scenario results where only one evolution occurs, the improvements will most likely increase for multiple rule evolutions using the same materializations. That is, given that rule evolution can be several orders of magnitude faster than running ER from scratch, further rule evolutions will cost a negligible amount of runtime compared to the naïve approach.

7.7 Without the properties

So far, we have only studied scenarios where one or more of the properties needed for our rule evolution techniques held. We now consider a scenario where the necessary properties do not hold. In this case, we need to use the naïve approach to get a correct answer. From our previous results, however, we know that the naïve approach can be very expensive compared with rule evolution. The alternatives are to fix the ER algorithm to satisfy one of the properties or to run one of our rule evolution algorithms even though we will not get correct answers. We investigate the latter case and see if we can still return ER results with minimum loss in accuracy.

We experiment on two ER algorithms that do not satisfy the $\mathcal{RM}$ property and thus cannot use Algorithm 1: the ME and HC_{DC} algorithms. While the ME algorithm is still $\mathcal{GI}$ and can thus use Algorithm 2, there is no runtime benefit because in ME all the records in P_i are extracted and sorted again regardless of the clusters in P_i.

To measure accuracy, we compare a rule evolution algorithm result with the corresponding result of the naïve approach. We consider all the records that merged into an output cluster to be identical to each other. For instance, if the clusters $\{r\}$ and $\{s\}$ merged into $\{r, s\}$ and then merged with $\{t\}$ into $\{r, s, t\}$, all three records r, s, t are considered to be the same. Suppose that the correct answer A contains the set of record pairs that match for the naïve solution while set B contains the matching pairs for the rule evolution algorithm. Then, the precision Pr is $\frac{|A\cap B|}{|B|}$ while the recall Re is $\frac{|A\cap B|}{|A|}$. Using Pr and Re, we compute the F_1-measure, which is defined as $\frac{2\times Pr\times Re}{Pr+Re}$, and use it as our accuracy metric.

Table 9 shows the runtime and accuracy results of running Algorithm 1 as the rule evolution algorithm on datasets that fit in memory. The columns show the dataset used and the number of records resolved. The top two rows of data show the runtimes for the naïve approach. The middle two rows of data show the runtime improvements of rule evolution compared with the naïve approaches. Each runtime improvement is computed by dividing the naïve approach runtime by the rule evolution runtime (not including the materialization costs). Overall, the runtime of ME improves by 1.67x to 5.53x while the runtime of HC_{DC} improves by 501x to 2386x. The bottom two rows of data show the accuracy values of each ER result compared with the correct result produced by the naïve aproach. The accuracy results are near-perfect for the ME algorithm while being at least 0.85 for HC_{DC}. The experiments show that rule evolution may produce highly accurate ER results even if the ER algorithms do not satisfy any property while still significantly enhancing the runtime performance of rule evolution.

7.8 Data evolution

We now evaluate how data evolution can speed up ER. Table 10 shows the ER runtimes for resolving 500 additional shopping (hotel) records after 3K random shopping (hotel)

Table 9 Runtime and accuracy results for ER algorithms without the properties

ER alg.	Sh1K	Sh2K	Sh3K	Ho1K	Ho2K	Ho3K
ER algorithm runtime (seconds)						
ME	0.094	0.162	0.25	0.015	0.033	0.051
HC_{DC}	8.08	39.2	105	5.51	28.1	73.57
Ratio of ER algorithm runtime to rule evolution time						
ME	5.53	5.23	5.43	1.67	2.06	2.04
HC_{DC}	674	1509	2386	501	879	1115
F_1 accuracy of rule evolution						
ME	0.94	0.95	0.97	1.0	1.0	0.997
HC_{DC}	0.93	0.86	0.85	1.0	0.999	0.999

Table 10 Data evolution runtime (seconds) when resolving 500 additional records. (An asterisk means that the resolution result is approximate. See text.)

ER alg.	Sh3K Naïve	Data Evol. Naïve	Ho3K	Data Evol.
SN	0.303	0.303	0.095	0.095
HC_B	22.1	5.81	8.85	2.14
HC_{BR}	78.8	28.5	5.23	1.63
HC_{DS}	161	100	101	58.8
ME	0.268	0.268	0.134	0.134
HC_{DC}	149	103 (*)	100	55.7 (*)

records have been resolved. The naïve approach is to run ER on 3K + 500 = 3,500 records while the data evolution approach resolves the 500 records starting from the previous ER result of the 3K records. As a result, data evolutions for the HC_B, HC_{BR}, HC_{DS}, and HC_{DC} algorithms outperform the naïve approaches by 1.5–4.1x. However, the SN and ME algorithms do not have better runtimes for data evolution because they re-sort all the records before resolving them (i.e., effectively starting ER from scratch). Notice that, in the worst case, data evolution has the same performance as starting ER from scratch. An interesting future enhancement is to add algorithm-specific optimizations (e.g., caching record comparison results) to make data evolution more efficient.

Recall that the $\mathcal{GI}$ property does not hold for the HC_{DC} algorithm. (The property holds for all the other algorithms of Table 10.) Thus, the HC_{DC} result may not be 100 % correct. In practice, one may still want to do incremental resolution even with non-$\mathcal{GI}$ algorithms to obtain the performance gains, even at a slight loss of accuracy. In our test scenario, for HC_{DC}, the accuracies of the data evolution ER results compared with the naïve (but correct) results are 0.91 and 0.999 for the shopping and hotel datasets, respectively. In general, the degree of such divergence depends on both the ER algorithm and the dataset. If the application cannot tolerate any error, then the ER algorithm needs to be modified to satisfy the $\mathcal{GI}$ property.

8 Related work

Entity resolution has been studied under various names including record linkage [25], merge/purge [16], deduplication [26], reference reconciliation [9], object identification [27], and others (see [8,10,13,31] for recent surveys). Entity resolution involves comparing records and determining whether they refer to the same entity or not. Most of the works fall into one of the ER models we consider: match-based clustering [3,16], distance-based clustering [4,21], or pairs ER [2,26,31]. While the ER literature focuses on improving the accuracy or runtime performance of ER, they usually assume that the logic and data are fixed during resolution. To the best of our knowledge, our work is the first to consider the ER result update problem when the ER logic itself changes.

Materializing ER results is related to the topic of query optimization using materialized views, which has been studied extensively in the database literature [7,11]. The focus of materialized views, however, is on optimizing the execution of SQL queries. In comparison, our work solves a similar problem for comparison rules that are Boolean or distance functions. Our work is also related to constructing data cubes [15] in data warehouses where each cell of a data cube is a view consisting of an aggregation (e.g., sum, average, count) of interests like total sales. More recently, materialization lists [20] have been used to enhance ER scalability. Each materialization list ranks pairs of records based on their similarity according to a field. In comparison, our rule evolution techniques store the ER results of comparison rules. Nonetheless, we believe our rule evolution techniques can improve by using techniques from the literature above. For example, deciding which combinations of conjuncts to materialize is related to the problem of deciding which views to materialize.

Data evolution is related to the problem of clustering data streams. Charikar et al. [5] propose incremental clustering algorithms that minimize the maximum cluster diameter given a stream of records. Aggarwal et al. [1] propose the

CluStream algorithm, which views a stream as a changing process over time and provides clustering over different time horizons in an evolving environment. Our work complements the above techniques by proposing a formal property ($\mathcal{GI}$) that guarantees the correctness of data evolution.

One of the recent challenges in information integration research is called Holistic Information Integration [14] where both schema and data issues are addressed within a single integration framework. For example, schema mapping can help with understanding the data and thus with ER while ER could also provide valuable information for schema mapping. Hence, schema mapping and ER can mutually benefit each other in an iterative fashion. While our work does not address the schema mapping problem, we provide a framework for iteratively updating ER results when the comparison logic (related to the schema) changes or new data are added.

This paper significantly extends a previous conference publication [29] by studying rule and data evolution for three general ER models – two based on clustering records and one based on finding matching pairs of records.

9 Conclusion

In most ER scenarios, the logic and data for resolving records evolve over time, as the application itself evolves and as the expertise for comparing records improves. In this paper, we have explored a fundamental question: When and how can we base a resolution on a previous result as opposed to starting from scratch? We have answered this question for clustering-based and pairs ER in two commonly used contexts, record comparisons based on Boolean predicates, and record comparisons based on distance (or similarity) functions. We identified two properties of ER algorithms, rule monotonic, and context free (in addition to order independent and general incremental) that can significantly reduce runtime at evolution time. We also categorized several popular ER algorithms according to the four properties.

In some cases, computing an ER result with new rules or data can be much faster if certain partial results are materialized when the original ER result (with the old rule) is computed. We studied how to take advantage of such materializations, and how they could be computed efficiently by piggybacking the work on the original ER computation.

Our experimental results focused on rule and data evolution for clustering ER and evaluated the cost of both materializations and the evolution itself (computing the new ER result), as compared to a naïve approach that computed the new result from scratch. We considered a variety of popular ER algorithms (each having different properties), two datasets, and different predicate strictness. The results illustrate realistic cases where materialization costs are relatively low, and evolution can be done extremely quickly.

Overall, we believe our analysis and experiments provide guidance for the ER algorithm designer. The experimental results show the potential gains, and if these gains are attractive in an application scenario, our properties help us design algorithms that can achieve such gains.

10 Proofs

10.1 Rule evolution

Lemma 1 *If* $\forall i,\ P \leq P_i$ *then* $P \leq \bigwedge P_i$.

Proof We can prove by induction on the number of partitions that are combined with the meet operation. □

Lemma 2 *For an* $\mathcal{RM}$ *algorithm,* $\forall P_o \in \bar{E}(P_i, B_2),\ P_o \leq M$.

Proof We first use the $\mathcal{RM}$ property to prove that $\forall P_o \in \bar{E}(P_i, B_2), P_o \leq M$. For each $conj \in Conj(B_1) \cap Conj(B_2)$, $B_2 \leq conj$. Hence, by $\mathcal{RM}$, $\forall P_o^1 \in \bar{E}(P_i, B_2)$ and $\forall P_o^2 \in \bar{E}(P_i, conj)$, $P_o^1 \leq P_o^2$. Since $M = \bigwedge_{conj \in Conj(B_1) \cap Conj(B_2)} E(P_i, conj)$, we conclude that $\forall P_o \in \bar{E}(P_i, B_2), P_o \leq M$ using Lemma 1. □

Lemma 3 *Suppose we have an algorithm that is* $\mathcal{RM}$ and $\mathcal{CF}$, *an initial partition* P_i, *and two rules* B_1, B_2 *with the conjuncts* $Conj(B_1), Conj(B_2)$, *respectively. Let* $M = \bigwedge_{conj \in Conj(B_1) \cap Conj(B_2)} E(P_i, conj)$. *For any* $W \subseteq M$, $E(\text{IN}(P_i, W), B_2) \leq W$.

Proof We use $\mathcal{CF}$ to prove that for any $W \subseteq M$, $E(\text{IN}(P_i, W), B_2) \leq W$. To avoid confusion with the initial set of clusters P_i, we use the symbol P_i' instead of P_i when using Definition 5. We satisfy the first two conditions in Definition 5 by setting $P = \text{IN}(P_i, W)$ and $P_i' = P_i$. The first condition, $P \subseteq P_i'$, is satisfied because $\text{IN}(P_i, W)$ is a subset of P_i by definition. The second condition, $\forall P_o \in \bar{E}(P_i', B_2), P_o \leq \{\bigcup_{c \in P} c, \bigcup_{c \in P_i' - P} c\}$, is satisfied because we know by Lemma 2 that $\forall P_o \in \bar{E}(P_i, B_2), P_o \leq M$. Also, we know that $W \subseteq M$. Thus, for $P_o^1 = E(P, B_2)$ and $P_o^2 = E(P_i' - P, B_2)$, $P_o^1 \cup P_o^2 = E(P_i', B_2)$. Since $E(P_i', B_2) \leq M$, $P_o^1 = E(\text{IN}(P_i, W), B_2) \leq W$, which also implies that $E(\text{IN}(P_i, W), B_2) \leq M$. □

Lemma 4 *Given the same setup as in Lemma 3, let* $Y \subseteq M$ *and* $Z \subseteq M$ *such that* $Y \cap Z = \emptyset$ *and* $Y \cup Z = W$ (*note*: $W \subseteq M$). *Let* $Q = E(\text{IN}(P_i, W), B_2)$ (*there is only one solution*). *Then* $Q \leq \{\bigcup_{c \in Y} c, \bigcup_{c \in Z} c\}$.

Proof Suppose that $Q \not\leq \{\bigcup_{c \in Y} c, \bigcup_{c \in Z} c\}$. Then, a cluster in Q must have one cluster from Y and one from Z. Since $Q \leq M$ (by Lemma 3), however, we arrive at a contradiction. □

Proposition 1 *Algorithm* 1 *correctly returns a partition* $P_o \in \bar{E}(P_i, B_2)$.

Proof We use $\mathcal{CF}$ to prove that $P_o = \bigcup_{c \in M} E(\{c' \in P_i | c' \subseteq c\}, B_2) \in \bar{E}(P_i, B_2)$. Suppose that $M = \{c_1, \ldots, c_{|M|}\}$. We omit the B_2 from any expression $E(P, B_2)$ for brevity (B_1 is not used in this proof). To avoid confusion with the initial set of clusters P_i, we use the symbol P_i' instead of P_i when using Definition 5 later in this proof. We define the following notation: $\alpha(k) = \bigcup_{c \in \{c_1, \ldots, c_k\}} E(\text{IN}(P_i, c))$ and $\beta(k) = \bigcup_{c \in M - \{c_1, \ldots, c_k\}} \text{IN}(P_i, c)$. Our goal $\bigcup_{c \in M} E(\{c' \in P_i | c' \subseteq c\}, B_2) \in \bar{E}(P_i, B_2)$ can thus be written as $\alpha(|M|) \in \bar{E}(P_i)$. To prove that $\alpha(|M|) \in \bar{E}(P_i)$, we first prove a more general statement: any $\alpha(k) \cup E(\beta(k)) \in \bar{E}(P_i)$ for $k \in \{0, \ldots, |M|\}$. Clearly, if our general statement holds, we can show that $\alpha(|M|) \in \bar{E}(P_i)$. We use induction on the number of partitions k processed in isolation.

Base case: $k = 0$. Then any $\alpha(0) \cup E(\beta(0)) = E(P_i) \in \bar{E}(P_i)$.

Induction: Suppose the equation above holds for $k = n$ (i.e., any $\alpha(n) \cup E(\beta(n)) \in \bar{E}(P_i)$). We want to show that the equation also holds when $k = n + 1$ where $n + 1 \le |M|$.

We first show that $E(\text{IN}(P_i, c_{n+1})) \cup E(\beta(n + 1)) = E(\beta(n))$ using $\mathcal{CF}$. We satisfy the first two conditions of Definition 5 by setting $P = \text{IN}(P_i, c_{n+1})$ and $P_i' = \beta(n)$. The first condition, $P \subseteq P'$, is satisfied because $\text{IN}(P_i, c_{n+1})$ is a subset of $\beta(n)$ by definition. The second condition, $\forall P_o \in \bar{E}(P_i'), P_o \le \{\bigcup_{c \in P} c, \bigcup_{c \in P_i' - P} c\}$, is satisfied by Lemma 4 by setting $Y = \{c_{n+1}\}$ and $Z = \{c_{n+2}, \ldots, c_{|M|}\}$. Hence, for $P_o^1 = E(P)$ and $P_o^2 = E(P_i' - P)$, $P_o^1 \cup P_o^2 = E(\text{IN}(P_i, c_{n+1})) \cup E(\beta(n + 1)) = E(\beta(n))$.

Now $\alpha(n+1) \cup E(\beta(n+1)) = \alpha(n) \cup E(\text{IN}(P_i, c_{n+1})) \cup E(\beta(n + 1))$, which is equal to some result $\alpha(n) \cup E(\beta(n))$. Using the induction hypothesis, we know that any $\alpha(n) \cup E(\beta(n)) \in \bar{E}(P_i)$, so $\alpha(n + 1) \cup E(\beta(n + 1)) \in \bar{E}(P_i)$ as well, which proves our induction step. □

Proposition 2 *The complexity of Algorithm* 1 *is* $O(c \times |S| + \frac{|S|^c}{|S|^{c-1}+z^c} \times g(\frac{|P_i| \times (|S|^{c-1}+z^c)}{|S|^c}, \frac{|S|}{|P_i|}))$ *where* S *is the set of records in the input partition of records* P_i, c *is the number of common conjuncts between* B_1 and B_2, z *is the average cluster size for any partition produced by a conjunct, and* $g(N, A)$ *is the complexity of the ER algorithm* E *for an input partition containing* N *clusters with an average size of* A *records.*

Proof The complexity of Algorithm 1 can be computed by adding the cost for meeting partitions of the common conjuncts (Step 3) and the cost for running ER on the clusters in M (Step 4). In Step 3, we perform $c - 1$ meets, which takes about $O(c \times |S|)$ time where the meet operation can be run in $O(|S|)$ time [23]. Given a record r, the probability of some other record s clustering with r is $\frac{z-1}{|S|-1}$ because each cluster has an average size of z. The probability for s to be in the same cluster with r for all the c meeting partitions is thus $(\frac{z-1}{|S|-1})^c$, assuming that all conjuncts cluster records independently. As a result, the expected number of records to be clustered with r in M is $(|S| - 1) \times (\frac{z-1}{|S|-1})^c$. Hence, the average cluster size of M is $1 + (|S| - 1) \times (\frac{z-1}{|S|-1})^c \approx 1 + \frac{z^c}{|S|^{c-1}}$ records. The expected number of clusters in M is thus approximately $\frac{|S|}{1+\frac{z^c}{|S|^{c-1}}} = \frac{|S|^c}{|S|^{c-1}+z^c}$. Each $\{c' \in P_i | c' \subseteq c\}$ (where $c \in M$) has on average $\frac{|P_i|}{\frac{|S|^c}{|S|^{c-1}+z^c}} = \frac{|P_i| \times (|S|^{c-1}+z^c)}{|S|^c}$ clusters of P_i where the average size of each cluster in P_i is $\frac{|S|}{|P_i|}$. The complexity of Step 4 is thus $O(\frac{|S|^c}{|S|^{c-1}+z^c} \times g(\frac{|P_i| \times (|S|^{c-1}+z^c)}{|S|^c}, \frac{|S|}{|P_i|}))$. Hence, the total algorithm complexity is $O(c \times |S| + \frac{|S|^c}{|S|^{c-1}+z^c} \times g(\frac{|P_i| \times (|S|^{c-1}+z^c)}{|S|^c}, \frac{|S|}{|P_i|}))$. □

Proposition 3 *Algorithm* 2 *correctly returns an ER result* $P_o \in \bar{E}(P_i, B_2)$.

Proof Suppose $M = \{c_1, c_2, \ldots, c_{|M|}\}$. For this proof, we denote $\{c' \in P_i | c' \subseteq c\}$ as $\text{IN}(P_i, c)$. We also omit B_2 from each $E(P, B_2)$ expression for brevity (B_1 is not used in this proof). We define the following notations: $\alpha(k) = \bigcup_{c \in M - \{c_1, \ldots, c_k\}} E(\text{IN}(P_i, c))$ and $\beta(k) = \bigcup_{c \in \{c_1, \ldots, c_k\}} \text{IN}(P_i, c)$. To avoid confusion with the initial set of clusters P_i, we use P_i' instead of P_i when using Definition 6 later in this proof. To prove that $P_o = E(\alpha(0)) \in \bar{E}(P_i) = \bar{E}(\beta(|M|))$, we prove the more general statement that $P_o \in \bar{E}(\alpha(k) \cup \beta(k))$ for $k \in \{0, \ldots, |M|\}$. Clearly, if our general statement holds, we can show that $P_o \in \bar{E}(\beta(|M|)) = \bar{E}(P_i)$ by setting $k = |M|$.

Base case: We set $k = 0$. Then $P_o = E(\alpha(0)) \in \bar{E}(\alpha(0)) = \bar{E}(\alpha(0) \cup \beta(0))$.

Induction: Suppose that our statement holds for $k = n$, i.e., $P_o = E(\alpha(0)) \in \bar{E}(\alpha(n) \cup \beta(n))$. We want to show that the same expression holds for $k = n + 1$ where $n + 1 \le |M|$. We use the $\mathcal{GI}$ property by setting $P = \text{IN}(P_i, c_{n+1})$ and $P_i' = \alpha(n + 1) \cup \beta(n + 1)$. The first condition $P \subseteq P_i'$ is satisfied because $\beta(n + 1)$ contains P. We then set $P_o^1 = E(P) = E(\text{IN}(P_i, c_{n+1}))$ and $P_o^2 = E(P_o^1 \cup (P_i' - P)) = E(E(\text{IN}(P_i, c_{n+1})) \cup \alpha(n + 1) \cup \beta(n)) = E(\alpha(n) \cup \beta(n))$. The $\mathcal{GI}$ property tells us that $P_o^2 \in \bar{E}(P_i') = \bar{E}(\alpha(n + 1) \cup \beta(n + 1))$. Thus, any $E(\alpha(n) \cup \beta(n)) \in \bar{E}(\alpha(n + 1) \cup \beta(n + 1))$. Using our induction hypothesis, we conclude that $P_o = E(\alpha(0)) \in \bar{E}(\alpha(n) \cup \beta(n)) \subseteq \bar{E}(\alpha(n+1) \cup \beta(n+1))$. □

Proposition 4 *If the ER algorithm* E *is* $\mathcal{GI}$, *then given a previous ER result* P_o *(produced from* P_i*) and a new partition* P_i', *the incremental data algorithm correctly produces an ER result of* $P_i \cup P_i'$.

Proof Suppose that $P_o \in \bar{E}(P_i, B)$. Also say that $P_o^2 \in \bar{E}(P_o \cup P_i, B) = \bar{E}(P_o \cup (P_i \cup P_i' - P_i), B)$. Then, by the

$\mathcal{GI}$ property, $P_o^2 \in \bar{E}(P_i \cup P_i', B)$, so P_o^2 is an ER result that can be produced from $P_i \cup P_i'$. □

10.2 Variations

Proposition 5 *The Join-Based algorithm is* $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{OI}$, *and* $\mathcal{GI}$.

Proof The proof is similar to that of Proposition 15. □

Proposition 6 *Algorithm* 3 *correctly returns the set of record pairs* $J_o \in \bar{J}_E(R, R, B_2)$.

Proof The proof is similar to that of Proposition 1. □

Proposition 7 *If a pairs ER algorithm E is* $\mathcal{GI}$, *then given a previous ER result* J_o *(produced from R) and a new set of records S, the incremental data algorithm correctly produces an ER result of* $R \cup S$.

Proof The proof is similar to that of Proposition 4. □

10.3 ER algorithms and their properties

Proposition 8 *The SN algorithm is* $\mathcal{RM}$, but not $\mathcal{CF}$.

Proof We prove that the SN algorithm is $\mathcal{RM}$. Given any partition P and two comparison rules B_1 and B_2 such that $B_1 \leq B_2$, the set of pairs of matching records found by B_1 is clearly a subset of that found by B_2, during the first phase of the SN algorithm. As a result, the transitive closure of the matching pairs by B_1 refines the transitive closure result of B_2 (i.e., $P_o^1 \leq P_o^2$).

We prove that the SN algorithm is not $\mathcal{CF}$ using a counter example. Suppose that the input partition is $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$, and we sort the records in P_i by their record IDs (e.g., the record r_2 has the ID of 2) into the sorted list $[r_1, r_2, r_3]$. Given the comparison rule B, suppose that only the pair r_1 and r_3 match with each other. Using a window size of 2, we do not identify any matching pairs of records because r_1 and r_3 are never in the same window according to the sorted list. Hence, $E(P_i, B) = \{\{r_1\}, \{r_2\}, \{r_3\}\}$. To apply the $\mathcal{CF}$ property, we set $P = \{\{r_1\}, \{r_3\}\}$ and $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$. The first two conditions in Definition 5 are satisfied because $P \subseteq P_i$ and $\forall P_o \in \bar{E}(P_i, B) = \{\{\{r_1\}, \{r_2\}, \{r_3\}\}\}$, $P_o \leq \{\bigcup_{c \in P}, \bigcup_{c \in P_i}\} = \{\{r_1, r_3\}, \{r_2\}\}$. Also, $E(P, B)$ is always $\{\{r_1, r_3\}\}$ (because r_1 and r_3 surely match being in the same window) and $E(P_i - P, B)$ is always $\{\{r_2\}\}$. However, $E(P, B) \cup E(P_i - P, B) = \{\{r_1, r_3\}, \{r_2\}\} \not\subseteq \bar{E}(P_i, B) = \{\{\{r_1\}, \{r_2\}, \{r_3\}\}\}$, violating the $\mathcal{CF}$ property. □

Proposition 9 *The* HC_B *algorithm is* $\mathcal{CF}$, *but not* $\mathcal{RM}$.

Proof We prove that the HC_B algorithm is $\mathcal{CF}$. Given the four partitions P, P_i, P_o^1, P_o^2 of Definition 5, suppose that $P_o^1 \cup P_o^2 \notin \bar{E}(P_i, B)$. We then prove that the four conditions of Definition 5 cannot all be satisfied at the same time. We first assume that the first, third, and fourth conditions are satisfied. That is, $P \subseteq P_i$, $P_o^1 \in \bar{E}(P, B)$, and $P_o^2 \in \bar{E}(P_i - P, B)$. Now suppose when deriving $E(P_i, B)$ that we "replay" all the merges among the clusters of P that were used to derive P_o^1 and then "replay" all the merges among the clusters of $P_i - P$ that were used to derive P_o^2. We thus arrive at the state $P_o^1 \cup P_o^2$ and can further merge any matching clusters until no clusters match to produce a possible ER result in $\bar{E}(P_i, B)$. Since $P_o^1 \cup P_o^2 \notin \bar{E}(P_i, B)$, there must have been new merges among clusters in P_o^1 and P_o^2. Since none of the clusters within P_o^1 or clusters within P_o^2 match with each other, we know that there must have been at least one more merge between a cluster in P_o^1 and a cluster in P_o^2 when deriving $E(P_i, B)$. As a result, the second condition cannot hold because there exists an ER result $P_o \in \bar{E}(P_i, B)$ such that $P_o \not\leq \{\bigcup_{c \in P} c, \bigcup_{c \in P_i - P} c\}$ because there exists a cluster in P_o that contains records in P as well as $P_i - P$. Hence, we have proved that all four conditions can never be satisfied if $P_o^1 \cup P_o^2 \notin \bar{E}(P_i, B)$.

We prove that the HC_B algorithm is not $\mathcal{RM}$ using a counter example. Consider the example, we used for illustrating the HC_B algorithm where $\bar{E}(P_i, B)$ was $\{\{\{r_1, r_2\}, \{r_3\}\}, \{\{r_1\}, \{r_2, r_3\}\}\}$. Now without having to define a new Boolean comparison rule, by setting $B_1 = B, B_2 = B, P_o^1 = \{\{r_1, r_2\}, \{r_3\}\}$, and $P_o^2 = \{\{r_1\}, \{r_2, r_3\}\}$, we see that $P_o^1 \not\leq P_o^2$ (although $B_1 \leq B_2$), contradicting the $\mathcal{RM}$ property. This result suggests that any ER algorithm that can produce more than one possible partition given any P_i and B is not $\mathcal{RM}$. We show in Proposition 13 the equivalent statement that any ER algorithm that is $\mathcal{RM}$ always returns a unique solution. □

Lemma 5 *Two records r and s are connected under B and* P_i *if and only if r and s are in the same cluster in* $P_o \in \bar{E}(P_i, B)$ *using the* HC_{BR} *algorithm.*

Proof Suppose that r and s are in the same cluster in P_o. If r and s are in the same cluster in P_i, then r and s are trivially connected under B and P_i. Otherwise, there exists a sequence of merges of the clusters in P_i that grouped r and s together. If two clusters c_a and c_b in P_i merge where $r \in c_a$ and $s \in c_b$, then r and s are connected because there is at least one pair of records $r' \in c_a$ and $s' \in c_b$ such that r' and s' match (i.e., $B(r', s') = \texttt{true}$), and r is connected to r' while s is connected to s'. Furthermore, we can prove that any record in $c_a \cup c_b$ is connected with any record in a cluster c_c that merges with $c_a \cup c_b$ using a similar argument: We know there exists a pair of records $r' \in c_a \cup c_b$ and $s' \in c_c$ that match with each other, and r is connected to r' while s is connected to s', which implies that r and s are connected under B and P_i. By repeatedly applying the same argument, we can prove that any r and s are connected if they end up in the same cluster in P_o.

Conversely, suppose that r and s are connected as the sequence $[r_1(=r), \ldots, r_n(=s)]$ under B and P_i. If r and s are in the same cluster in P_i, they are already clustered together in P_o. Otherwise, all the clusters that contain $r_1, \ldots, r_n$ eventually merge together according to the HC_{BR} algorithm, clustering r and s together in P_o. □

Proposition 10 *The HC_{BR} algorithm always returns a unique solution.*

Proof Suppose that HC_{BR} produces two different output partitions for a given partition P_i and comparison rule B, i.e., $\bar{E}(P_i, B) = \{P_o^1, P_o^2, \ldots\}$. Then, there must exist two records r and s that have merged into the same cluster according to one ER result, but not in the same cluster for the other ER result. Suppose that r and s are in the same cluster in P_o^1, but in separate clusters in P_o^2. Since r and s are clustered together in P_o^1, they are connected by Lemma 5. Hence, r and s must also be clustered in P_o^2 again by Lemma 5, contradicting our hypothesis that they are in different clusters in P_o^2. Hence, HC_{BR} always returns a unique partition. □

Proposition 11 *The HC_{BR} algorithm is both $\mathcal{RM}$ and $\mathcal{CF}$.*

Proof The HC_{BR} algorithm is $\mathcal{CF}$ because the HC_B algorithm already is $\mathcal{CF}$. To show that the HC_{BR} algorithm also is $\mathcal{RM}$, suppose that $B_1 \leq B_2$. Then, all the clusters that match according to B_1 also match according to B_2. Hence, for any $P_o^1 \in \bar{E}(P_i, B_1)$, we can always construct an ER result $P_o^2 \in \bar{E}(P_i, B_2)$ (which is unique by Proposition 10) where $P_o^1 \leq P_o^2$ by performing the exact same merges done for P_o^1 and then continuing to merge clusters that still match according to B_2 until no clusters match according to B_2. □

Proposition 12 *The ME algorithm does not satisfy $\mathcal{RM}$ or $\mathcal{CF}$.*

Proof We prove that the ME algorithm is not $\mathcal{RM}$ using a counter example. Suppose that the input partition is $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$, and we sort the records by their record IDs (e.g., the record r_2 has the ID of 2) into the sorted list of records $[r_1, r_2, r_3]$. Suppose that $B_1(r_1, r_3) = \texttt{true}$, but $B_1(r_1, r_2) = \texttt{false}$ and $B_1(r_2, r_3) = \texttt{false}$. Compared with B_1, the only difference of B_2 is that $B_2(r_2, r_3) = \texttt{true}$. Clearly, $B_1 \leq B_2$. Using a queue size of 2, $E(P_i, B_1)$ returns $\{\{r_1, r_3\}, \{r_2\}\}$ only because r_1 and r_2 are inserted into the queue separately, and r_3 then merges with $\{r_1\}$. On the other hand, $E(P_i, B_2)$ returns $\{\{r_1\}, \{r_2, r_3\}\}$ because r_1 and r_2 are inserted into the queue separately, and r_3 matches with $\{r_2\}$ first. Since $E(P_i, B_1) = \{\{r_1, r_3\}, \{r_2\}\} \not\leq \{\{r_1\}, \{r_2, r_3\}\} = E(P_i, B_2)$, the $\mathcal{RM}$ property does not hold.

We prove that the ME algorithm is not $\mathcal{CF}$ using a counter example. Suppose that the input partition is $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$, and we sort the records by their IDs into the sorted list of records $[r_1, r_2, r_3]$. Suppose that $B(r_1, r_3) = \texttt{true}$, but $B(r_1, r_2) = \texttt{false}$ and $B(r_2, r_3) = \texttt{false}$. Using a queue size of 1, we do not identify any matching pairs because r_1 is never compared with r_3 because once r_2 enters the queue, $\{r_1\}$ is pushed out of the queue. Hence, $E(P_i, B)$ is always $\{\{r_1\}, \{r_2\}, \{r_3\}\}$. Using Definition 6, suppose we set $P = \{\{r_1\}, \{r_3\}\}$ and $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$. The first condition is satisfied because $P \subseteq P_i$. The second condition is satisfied because $\forall P_o \in \bar{E}(P_i, B), P_o = \{\{r_1\}, \{r_2\}, \{r_3\}\} \leq \{\bigcup_{c \in P} c, \bigcup_{c \in P_i - P} c\} = \{\{r_1, r_3\}, \{r_2\}\}$. Also, P_o^1 is always $\{\{r_1, r_3\}\}$ because r_3 matches with $\{r_1\}$ while $\{r_1\}$ is in the queue, and P_o^2 is always $\{\{r_2\}\}$. As a result, $P_o^1 \cup P_o^2 = \{\{r_1, r_3\}, \{r_2\}\} \notin \bar{E}(P_i, B) = \{\{\{r_1\}, \{r_2\}, \{r_3\}\}\}$, contradicting $\mathcal{CF}$. □

Proposition 13 *Any ER algorithm that is $\mathcal{RM}$ is also $\mathcal{OI}$.*

Proof Suppose that we are given an $\mathcal{RM}$ ER algorithm E, and the $\mathcal{OI}$ property does not hold. Then, there exists a partition P_i and comparison rule B such that $\bar{E}(P_i, B)$ contains at least two different partitions P_x and P_y. Without loss of generality, we assume that $P_x \not\leq P_y$. However, we violate Definition 4 by setting $B_1 = B, B_2 = B, P_o^1 = P_x$, and $P_o^2 = P_y$ because $B_1 \leq B_2$, but $P_o^1 = P_x \not\leq P_y = P_o^2$. Hence, E cannot satisfy the $\mathcal{RM}$ property, a contradiction. □

Proposition 14 *The ME algorithm is $\mathcal{OI}$ and $\mathcal{GI}$, but not $\mathcal{RM}$ or $\mathcal{CF}$.*

Proof Proposition 12 shows that ME does not satisfy $\mathcal{RM}$ or $\mathcal{CF}$. The ME algorithm is $\mathcal{OI}$ because it first sorts the records in P_i before resolving them with a sliding window and thus produces a unique solution. The ME algorithm is $\mathcal{GI}$ because $E(P_o^1 \cup (P_i - P), B)$ always returns the same result as $E(P_i, B)$. That is, ME extracts all the records from its input partition before sorting and resolving them, and $P_o^1 \cup (P_i - P)$ contains the exact same records as those in P_i (i.e., $\bigcup_{c \in P_o^1 \cup (P_i - P)} c = \bigcup_{c \in P_i} c$). □

Proposition 15 *The HC_{BR} algorithm is $\mathcal{RM}, \mathcal{CF}, \mathcal{GI}$, and $\mathcal{OI}$.*

Proof Proposition 11 shows that HC_{BR} is $\mathcal{RM}$ and $\mathcal{CF}$. The HC_{BR} algorithm is also $\mathcal{OI}$ by Proposition 10. To show that HC_{BR} is $\mathcal{GI}$, consider the four partitions P, P_i, P_o^1, P_o^2 of Definition 6, and suppose that the three conditions $P \subseteq P_i$, $P_o^1 \in \bar{E}(P, B)$, and $P_o^2 \in \bar{E}(P_o^1 \cup (P_i - P), B)$ hold. We show that $P_o^2 \in \bar{E}(P_i, B)$. Starting from P_i, P_o^2 is the result of "replaying" the merges used to produce P_o^1, and then "replaying" the merges used to produce a possible result of $E(P_o^1 \cup (P_i - P), B)$. Since no clusters in P_o^2 match with each other, P_o^2 is also a possible result for $E(P_i, B)$. Hence, $P_o^2 \in \bar{E}(P_i, B)$. □

Proposition 16 *The HC_B algorithm is $\mathcal{CF}$ and $\mathcal{GI}$, but not $\mathcal{OI}$ (and consequently not $\mathcal{RM}$).*

Proof Proposition 9 shows that HC_B is $\mathcal{CF}$. The proof that HC_B is $\mathcal{GI}$ is identical to the proof for HC_{BR} and is omitted.

The HC_B algorithm does not satisfy $\mathcal{OI}$ because, depending on the order of records compared, there could be several possible ER results. For example, given the input partition $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$ and the comparison rule B, suppose that $B(r_1, r_2) = \texttt{true}$ and $B(r_2, r_3) = \texttt{true}$, but $B(r_1, r_3) = \texttt{false}$. Also assume that, whenever we compare two clusters of records, we simply compare the records with the smallest IDs from each cluster using B. For instance, when comparing $\{r_1, r_2\}$ with $\{r_3\}$, we return the result of $B(r_1, r_3)$. Then depending on the order of clusters in P_i compared, the HC_B algorithm either produces $\{\{r_1, r_2\}, \{r_3\}\}$ or $\{\{r_1, r_2, r_3\}\}$. □

Proposition 17 *There exists an ER algorithm that is $\mathcal{RM}$ (and consequently $\mathcal{OI}$ as well), but not $\mathcal{GI}$ or $\mathcal{CF}$.*

Proof We define a variant of the SN algorithm called SN^2. Recall that the SN algorithm involves identifying the matching pairs of records by first sorting the records from P_i and then comparing records within the same sliding window. At the end, we produce a transitive closure of all the matching records. During the transitive closure, however, we define SN^2 to also consider all the records within the same cluster in the input partition P_i as matching. For example, suppose we have $P_i = \{\{r_1, r_2\}, \{r_3\}\}$ and a comparison rule B such that $B(r_1, r_3) = \texttt{true}$, but $B(r_1, r_2) = \texttt{false}$ and $B(r_2, r_3) = \texttt{false}$. Given a window size of 2, suppose that we sort the records in P_i by record ID into the list $[r_1, r_2, r_3]$. Then the original SN algorithm sorts will return $\{\{r_1\}, \{r_2\}, \{r_3\}\}$ because r_1 and r_3 are never compared in the same window. However, the new SN^2 algorithm will consider r_1 and r_2 as matching because they were in the same cluster in the input partition P_i. Hence, the result of SN^2 is $\{\{r_1, r_2\}, \{r_3\}\}$.

We prove that the SN^2 algorithm is $\mathcal{RM}$. Given any partition P and two comparison rules B_1 and B_2 such that $B_1 \leq B_2$, the set of pairs of matching records found by B_1 is clearly a subset of that found by B_2, during the first phase of the SN algorithm where we find all the matching pairs of records within the same sliding window at any point. In addition, the set of matching pairs of records identified from P_i is the same for both B_1 and B_2. As a result, the transitive closure of the matching pairs by B_1 refines the transitive closure result of B_2 (i.e., for any $P_o^1 \in \bar{E}(P_i, B_1)$ and $P_o^2 \in \bar{E}(P_i, B_2)$, $P_o^1 \leq P_o^2$).

We prove that the SN^2 algorithm is not $\mathcal{GI}$ using a counter example. Suppose that we have $P_i = \{\{r_1, r_2\}, \{r_3\}\}$ and a comparison rule B such that $B(r_1, r_3) = \texttt{true}$, but $B(r_1, r_2) = \texttt{false}$ and $B(r_2, r_3) = \texttt{false}$. Given a window size of 2, suppose that we sort the records in P_i by record ID into the list $[r_1, r_2, r_3]$. As a result, $\bar{E}(P_i, B) = \{\{\{r_1\}, \{r_2\}, \{r_3\}\}\}$ because r_1 and r_3 are never compared within the same window. Using Definition 6, suppose that we set $P = \{\{r_1\}, \{r_3\}\}$ and $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$. As a result, $P_o^1 = E(P, B) = \{\{r_1, r_3\}\}$ because r_1 and r_3 surely match being in the same window. Also, $P_o^2 = E(P_o^1 \cup (P_i - P), B) = E(\{\{r_1, r_3\}, \{r_2\}\}, B) = \{\{r_1, r_3\}, \{r_2\}\}$ because r_1 and r_3 are in the same cluster of the input partition $P_o^1 \cup (P_i - P)$. Since $E(P_i, B)$ is always $\{\{r_1\}, \{r_2\}, \{r_3\}\}$, $P_o^2 \notin \bar{E}(P_i, B) = \{\{\{r_1\}, \{r_2\}, \{r_3\}\}\}$. Hence, the $\mathcal{GI}$ property is not satisfied. (On the other hand, the original SN algorithm is $\mathcal{GI}$; see Proposition 19.)

To prove that SN^2 is not $\mathcal{CF}$ (in order to show that Fig. 2 correctly categorizes SN^2), we can directly use the proof of Proposition 8 that was used to show SN is not $\mathcal{CF}$. □

Proposition 18 *There exists an ER algorithm that is $\mathcal{CF}$ and $\mathcal{OI}$, but not $\mathcal{GI}$ or $\mathcal{RM}$.*

Proof We define a variant of the HC_B algorithm (called HC_B^2) where all matching clusters are merged, but only in a certain order. For example, we can define a total ordering between pairs of clusters where the clusters with the "smallest record IDs" are merged first. We first define the following notations: $MinID_1 = min\{min_{r\in c_1}\text{ID}(r), min_{r\in c_2}\text{ID}(r)\}$, $MaxID_1 = max\{min_{r\in c_1}\text{ID}(r), min_{r\in c_2}\text{ID}(r)\}$, $MinID_2 = min\{min_{r\in c_3}\text{ID}(r), min_{r\in c_4}\text{ID}(r)\}$, and $MaxID_2 = max\{min_{r\in c_3}\text{ID}(r), min_{r\in c_4}\text{ID}(r)\}$ where the ID(r) function returns the ID of r. We merge the pair of clusters (c_1, c_2) before the pair of matching clusters (c_3, c_4) if either $MinID_1 < MinID_2$ or both $MinID_1 = MinID_2$ and $MaxID_1 < MaxID_2$. For example, the matching clusters $\{r_3, r_9\}$ and $\{r_6, r_7\}$ merge before the matching clusters $\{r_4, r_8\}$ and $\{r_6, r_7\}$ because $MinID_1 = 3$, $MaxID_1 = 6$, $MinID_2 = 4$, $MaxID_2 = 6$ and thus $MinID_1 < MinID_2$. In general, we can use any total ordering of pairs of clusters. As a result of using the total ordering, the HC_B^2 algorithm always produces a unique ER result. For example, suppose that we have $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$ and the Boolean comparison rule B where $B(r_1, r_2) = \texttt{true}$ and $B(r_2, r_3) = \texttt{true}$, but $B(r_1, r_3) = \texttt{false}$. Also assume that, whenever we compare two clusters of records, we simply compare the records with the smallest IDs from each cluster using B. For instance, when comparing $\{r_1, r_2\}$ with $\{r_3\}$, we return the result of $B(r_1, r_3)$. Then the ER result $E(P_i, B) = \{\{r_1, r_2\}, \{r_3\}\}$. The clusters $\{r_1\}$ and $\{r_2\}$ merge before $\{r_2\}$ and $\{r_3\}$ because $MinID_1 = 1 < 2 = MinID_2$. Once $\{r_1\}$ and $\{r_2\}$ merge, $\{r_1, r_2\}$ does not match with $\{r_3\}$ because $B(r_1, r_3) = \texttt{false}$.

We show that the HC_B^2 algorithm is $\mathcal{CF}$. Given the four partitions P, P_i, P_o^1, P_o^2, suppose that the four conditions in Definition 5 are satisfied. That is, $P \subseteq P_i, \forall P_o \in \bar{E}(P_i, B), P_o \leq \{\bigcup_{c \in P} c\ ,\ \bigcup_{c \in P_i - P} c\}, P_o^1 \in \bar{E}(P, B)$, and $P_o^2 \in \bar{E}(P_i - P, B)$. Since HC_B^2 returns a unique solution, there is exactly one $P_o \in \bar{E}(P_i, B)$. Suppose that P_o was generated via a sequence of cluster merges $M_1, M_2, \ldots$ where each M involves a merge of two clusters. Since $P_o \leq \{\bigcup_{c \in P} c\ ,\ \bigcup_{c \in P_i - P} c\}$, we can split the sequence into merges $M_1^a, M_2^a, \ldots$ that only involve clusters in P and merges $M_1^b, M_2^b, \ldots$ that only involve clusters in $P_i - P$. We can run the first batch of merges $M_1^a, M_2^a, \ldots$ to produce a possible result of $E(P, B)$ and run the second batch of merges $M_1^b, M_2^b, \ldots$ to produce a possible result of $E(P_i - P, B)$. Since HC_B^2 returns a unique solution, both ER results $E(P, B)$ and $E(P_i - P, B)$ are in fact unique and thus are equal to P_o^1 and P_o^2, respectively. The union of P_o^1 and P_o^2 is equivalent to the result of running the merges $M_1, M_2, \ldots$, i.e., $E(P_i, B)$. Hence, $P_o^1 \cup P_o^2 \in \bar{E}(P_i, B)$.

The HC_B^2 algorithm is $\mathcal{OI}$ because the merges are done in a fixed order.

We show that the HC_B^2 algorithm does not satisfy $\mathcal{GI}$ using a counter example. Suppose that we have $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$ and the Boolean comparison rule B where $B(r_1, r_2) = \texttt{true}$ and $B(r_2, r_3) = \texttt{true}$, but $B(r_1, r_3) = \texttt{false}$. Also assume that, whenever we compare two clusters of records, we simply compare the records with the smallest IDs from each cluster using B. For instance, when comparing $\{r_1, r_2\}$ with $\{r_3\}$, we return the result of $B(r_1, r_3)$. Then, the ER result $E(P_i, B) = \{\{r_1, r_2\}, \{r_3\}\}$ because $\{r_1\}$ and $\{r_2\}$ merge first (before $\{r_2\}$ and $\{r_3\}$) and then $\{r_1, r_2\}$ does not match with $\{r_3\}$ because $B(r_1, r_3) = \texttt{false}$. However, in Definition 6 suppose that we set $P = \{\{r_2\}, \{r_3\}\}$ and $P_i = \{\{r_1\}, \{r_2\}, \{r_3\}\}$. Then $P_o^1 = E(P, B) = \{\{r_2, r_3\}\}$ because $\{r_2\}$ matches with $\{r_3\}$. Also $P_o^2 = E(P_o^1 \cup (P_i - P), B) = E(\{\{r_2, r_3\}, \{r_1\}\}, B) = \{\{r_1, r_2, r_3\}\}$ because $\{r_2, r_3\}$ and $\{r_1\}$ match. Since $E(P_i, B)$ is always $\{\{r_1, r_2\}, \{r_3\}\}$, $P_o^2 \notin \bar{E}(P_i, B) = \{\{\{r_1, r_2\}, \{r_3\}\}\}$. Hence, the $\mathcal{GI}$ property is not satisfied.

We show that the HC_B^2 algorithm is not $\mathcal{RM}$ using a counter example. We use the same example in the previous paragraph where $E(P_i, B) = \{\{r_1, r_2\}, \{r_3\}\}$. Now suppose that we are given a stricter comparison rule B_1 where $B_1(r_2, r_3) = \texttt{true}$, but $B_1(r_1, r_2) = \texttt{false}$ and $B_1(r_1, r_3) = \texttt{false}$. Then $E(P_i, B_1) = \{\{r_1\}, \{r_2, r_3\}\}$ because only $\{r_2\}$ and $\{r_3\}$ match. Hence, although $B_1 \leq B$, $E(P_i, B) = \{\{r_1, r_2\}, \{r_3\}\} \not\leq E(P_i, B_1) = \{\{r_1\}, \{r_2, r_3\}\}$, violating $\mathcal{RM}$. □

Proposition 19 *The SN algorithm is $\mathcal{RM}$ (and consequently $\mathcal{OI}$) and $\mathcal{GI}$, but not $\mathcal{CF}$.*

Proof Proposition 8 shows that SN is $\mathcal{RM}$ (and consequently $\mathcal{OI}$), but not $\mathcal{CF}$. We prove that the SN algorithm is $\mathcal{GI}$. Recall that SN extracts all records in the input partition before sorting and resolving them. Hence, the ER result is the same for different input partitions as long as they contain the same records. For example, $E(\{\{r_1, r_2\}, \{r_3\}\}, B) = E(\{\{r_1\}, \{r_2, r_3\}\}, B)$ because the two input partitions contain the same records r_1, r_2, and r_3. In Definition 6, we know that $E(P_o^1 \cup (P_i - P), B)$ always returns the same result as $E(P_i, B)$ because $P_o^1 \cup (P_i - P)$ contains the same records as those in P_i. Thus, for any $P_2 \in \bar{E}(P_o^1 \cup (P_i - P), B)$, $P_2 \in \bar{E}(P_b, B)$. □

Proposition 20 *The extended HC_{DS} algorithm is $\mathcal{RM}$, $\mathcal{CF}$, $\mathcal{GI}$, and $\mathcal{OI}$.*

Proof The proof is analogous to that of Proposition 15. □

Proposition 21 *The extended HC_{DC} algorithm is $\mathcal{CF}$, but not $\mathcal{RM}$, $\mathcal{OI}$, or $\mathcal{GI}$.*

Proof The proof that HC_{DC} is $\mathcal{CF}$ is an extension of the analogous proof in Proposition 18. We can prove that HC_{DC} does not satisfy the other properties using counter examples. □

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: VLDB, pp. 81–92 (2003)
2. Arasu, A., Götz, M., Kaushik, R.: On active learning of record matching packages. In: SIGMOD Conference, pp. 783–794 (2010)
3. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. **18**(1), 255–276 (2009)
4. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. TKDD 1(1) (2007)
5. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental clustering and dynamic information retrieval. In: STOC, pp. 626–635 (1997)
6. Chaudhuri, S., Ganti, V., Motwani, R.: Robust identification of fuzzy duplicates. In: Proc. of ICDE. Tokyo, Japan (2005)
7. Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K.: Optimizing queries with materialized views. In: ICDE, pp. 190–200 (1995)
8. Christen, P.: Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Data-Centric Systems and Applications. Springer, Berlin (2012)
9. Dong, X., Halevy, A., Madhavan, J.: Reference reconciliation in complex information spaces. SIGMOD, In (2005)
10. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
11. Goldstein, J., Larson, P.Å.: Optimizing queries using materialized views: A practical, scalable solution. In: SIGMOD Conference, pp. 331–342 (2001)
12. Gower, J.C., Ross, G.J.S.: Minimum spanning trees and single linkage cluster analysis. Applied Statistics **18**(1), 54–64 (1969)
13. Gu, L., Baxter, R., Vickers, D., Rainsford, C.: Record linkage: Current practice and future directions. Tech. Rep. 03/83, CSIRO Mathematical and, Information Sciences (2003)
14. Haas, L.M., Hentschel, M., Kossmann, D., Miller, R.J.: Schema and data: A holistic approach to mapping, resolution and fusion in information integration. In: ER, pp. 27–40 (2009)

15. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. In: SIGMOD Conference, pp. 205–216 (1996)
16. Hernández, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: Proc. of ACM SIGMOD, pp. 127–138 (1995)
17. Indyk, P.: A small approximately min-wise independent family of hash functions. J. Algorithms **38**(1), 84–90 (2001)
18. Jain A.K.: Data clustering: 50 years beyond k-means. In: ECML/PKDD (1), pp. 3–4 (2008)
19. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: A review. ACM Comput. Surv. **31**(3), 264–323 (1999)
20. Lee, S., Lee, J., won Hwang, S.: Scalable entity matching computation with materialization. In: CIKM, pp. 2353–2356 (2011)
21. Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)
22. McCallum, A.K., Nigam, K., Ungar, L.: Efficient clustering of high-dimensional data sets with application to reference matching. In: Proc. of KDD, pp. 169–178. Boston, MA (2000)
23. Menestrina, D., Whang, S., Garcia-Molina, H.: Evaluating entity resolution results. PVLDB **3**(1), 208–219 (2010)
24. Monge, A.E., Elkan, C.: An efficient domain-independent algorithm for detecting approximately duplicate database records. In: DMKD, pp. 23–29 (1997)
25. Newcombe, H.B., Kennedy, J.M.: Record linkage: making maximum use of the discriminating power of identifying information. Commun. ACM **5**(11), 563–566 (1962)
26. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: Proc. of ACM SIGKDD. Edmonton, Alberta (2002).
27. Tejada, S., Knoblock, C.A., Minton, S.: Learning object identification rules for information integration. Information Systems Journal **26**(8), 635–656 (2001)
28. Whang, S.E., Benjelloun, O., Garcia-Molina, H.: Generic entity resolution with negative rules. VLDB J. **18**(6), 1261–1277 (2009)
29. Whang, S.E., Garcia-Molina, H.: Entity resolution with evolving rules. PVLDB **3**(1), 1326–1337 (2010)
30. Whang, S.E., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. In: SIGMOD Conference, pp. 219–232 (2009)
31. Winkler, W.: Overview of record linkage and current research directions. Tech. rep., Statistical Research Division, U.S. Bureau of the Census, Washington, DC (2006)