Generic Entity Resolution with Data Confidences

David Menestrina
dmenest@cs.stanford.edu

Omar Benjelloun
benjello@cs.stanford.edu

Hector Garcia-Molina
hector@cs.stanford.edu

Stanford University

ABSTRACT

We consider the *Entity Resolution* (*ER*) problem (also known as deduplication, or merge-purge), in which records determined to represent the same real-world entity are successively located and merged. Our approach to the ER problem is *generic*, in the sense that the functions for comparing and merging records are viewed as black-boxes. In this context, managing numerical confidences along with the data makes the ER problem more challenging to define (e.g., how should confidences of merged records be combined?), and more expensive to compute. In this paper, we propose a sound and flexible model for the ER problem with confidences, and propose efficient algorithms to solve it. We validate our algorithms through experiments that show significant performance improvements over naive schemes.

1. INTRODUCTION

When data from different sources is cleansed and integrated, often multiple input records refer to the same real-world entity, e.g., to the same customer, the same product or the same organization. *Entity resolution* (ER) identifies the records that refer (or are likely to refer) to the same entity, and merges these records. A merged record becomes a "composite" of the source records. In general, a merged record needs to be compared and possibly merged with other records, since the composition of information may now make it possible to identify new relationships. For instance, say record r_1 gives the name and driver's license of a person, while record r_2 gives an address and the same driver's license number. Say we merge r_1 and r_2 based on the matching driver's license. Now we have both a name and an address for this person, and this combined information may make it possible to connect this merged record with say r_3, containing a similar name and address. Note that neither r_1 nor r_2 may match with r_3, because they do not contain the combined information that the merged record has. Entity resolution is also known as deduplication and record linkage.

Often, numerical confidences (or data quality) play a role in entity resolution. For instance, the input records may come from unreliable sources, and have confidences or quality associated with them. The comparisons between records may also yield confidences that represent how likely it is that the records refer to the same real-world entity. Similarly, the merge process may introduce additional uncertainties, as it may not be certain how to combine the information from different records. In each application domain, the interpretation of the quality or confidence numbers may be different. For instance, a confidence number may represent a "belief" that a record faithfully reflects data from a real-world entity, or it may represent how "accurate" a record is.

Even though ER is a central problem in information integration, and even though confidences are often an integral part of resolution, relatively little is known about how to *efficiently* deal with confidences. Specifically, confidences may make the ER process computationally more expensive, as compared to a scenario where confidences are not taken into account. For instance, without confidences, the order in which records are merged may be unimportant, and this property can be used to find efficient ER strategies. However, confidences may make order critical. For instance, say we merge r_1 to r_2 and then to r_3, giving us a record r_{123}. Because r_1 and r_2 are "very similar", we may have a high confidence in the intermediate result, which then gives us high confidence in r_{123}. However, say we merge r_1 to r_3 and then to r_2, giving us record r_{132}. In this case, r_1 and r_3 may not be "that similar", leading t a lower confidence r_{132}. Records r_{123} and r_{132} may even have the same attributes, but may have different confidences because they were derived differently. Thus, ER must consider many more potential derivations of composite records.

Our goal in this paper is to explore ways to reduce the high computational costs of ER with confidences. We wish to achieve this goal without making too many assumptions about the confidence model and how confidences are computed when record are merged. Thus, we will use a generic *black-box model* for the functions that compare records, that merge records, and that compute new confidences. We will then postulate properties that these functions may have: if the properties hold, then efficient ER with confidences will be possible. If they do not hold, then one must run a more-general version of ER (as we will detail here). Since we use generic match and merge functions, the algorithms we present can be used in many domains. All that is required is to check what properties the match and merge functions have, and then to select the appropriate algorithm.

The contributions of this paper are the following:

CleanDB, Seoul, Korea, 2006

- We define a generic framework for managing confidences during entity resolution (Sections 2 and 3).
- We present Koosh, an algorithm for resolution when confidences are involved (Section 4).
- We present three improvements over Koosh that can significantly reduce the amount of work during resolution: domination, packages and thresholds. We identify properties that must hold in order for these improvements to be achievable (Sections 5, 6, and 7).
- We evaluate the algorithms and quantify the potential performance gains (Section 8).

2. MODEL

Each *record* r consists of a *confidence* $r.\mathcal{C}$ and a set of *attributes* $r.\mathcal{A}$. For illustration purposes, we can think of each attribute as a label-value pair, although this view is not essential for our work. For example, the following record may represent a person:

0.7 [name: "Fred", age: $\{45, 50\}$, zip: 94305]

In our example, we write $r.\mathcal{C}$ (0.7 in this example) in front of the attributes. (A record's confidences could simply be considered as one of its attributes, but here we treat confidences separately to make it easier to refer to them.) Note that the value for an attribute may be a set. In our example, the age attribute has two values, 45 and 50. Multiple values may be present in input records, or arise during integration: a record may report an age of 45 while another one reports 50. Some merge functions may combine the ages into a single number (say, the average), while others may decide to keep both possibilities, as shown in this example.

Note that we are using a single number to represent the confidence of a record. We believe that single numbers (in the range 0 to 1) are the most common way to represent confidences in the ER process, but more general confidence models are possible. For example, a confidence could be a vector, stating the confidences in individual attributes. Similarly, the confidence could include lineage information explaining how the confidence was derived. However, these richer models make it harder for application programmers to develop merge functions (see below), so in practice, the applications we have seen all use a single number.

Generic ER relies on two black-box functions, the *match* and the *merge* function, which we will assume here work on two records at a time:

- A match function $M(r, s)$ returns true if records r and s represent the same entity. When $M(r, s) = true$ we say that r and s match, denoted $r \approx s$.
- A merge function creates a composite record from two matching records. We represent the merge of record r and s by $\langle r, s\rangle$.

Note that the match and merge functions can use global information to make their decisions. For instance, in an initialization phase we can compute say the distribution of terms used in product descriptions, so that when we compare records we can take into account these term frequencies. Similarly, we can run a clustering algorithm to identify sets of input records that are "similar." Then the match function can consult these results to decide if records match. As new records are generated, the global statistics need to be updated (by the merge function): these updates can be done incrementally or in batch mode, if accuracy is not essential.

The pairwise approach to match and merge is often used in practice because it is easier to write the functions. (For example, ER products from IBM, Fair Isaac, Oracle, and others use pairwise functions.) For instance, it is extremely rare to see functions that merge more than two records at a time. To illustrate, say we want to merge 4 records containing different spellings of the name "Schwartz." In principle, one could consider all 4 names and come up with some good "centroid" name, but in practice it is more common to use simpler strategies. For example, we can just accumulate all spellings as we merge records, or we can map each spelling to the closest name in a dictionary of canonical names. Either approach can easily be implemented in a pairwise fashion.

Of course, in some applications pairwise match functions may not be the best approach. For example, one may want to use a set-based match function that considers a set of records and identifies the pair that should be matched next, i.e., $M(S)$ returns records $r, s \in S$ that are the best candidates for merging. Although we do not cover it here, we believe that the concepts we present here (e.g., thresholds, domination) can also be applied when set-based match functions are used, and that our algorithms can be modified to use set-based functions.

Pairwise match and merge are generally not arbitrary functions, but have some properties, which we can leverage to enable efficient entity resolution. We assume that the match and merge functions satisfy the following properties:

- *Commutativity:* $\forall r, s,\ r \approx s \Leftrightarrow s \approx r$ and if $r \approx s$ then $\langle r, s\rangle = \langle s, r\rangle$.
- *Idempotence:* $\forall r, r \approx r$ and $\langle r, r\rangle = r$.

We expect these properties to hold in almost all applications (unless the functions are not property implemented). In one ER application we studied, for example, the implemented match function was not idempotent: a record would not match itself if the fields used for comparison were missing. However, it was trivial to add a comparison for record equality to the match function to achieve idempotence. (The advantage of using an idempotent function will become apparent when we see the efficient options for ER.)

Some readers may wonder if merging two identical records should really give the same record. For example, say the records represent two observations of some phenomena. Then perhaps the merge record should have a higher confidence because there are two observations? The confidence would only be higher if the two records represent independent observations, not if they are identical. We assume that independent observations would differ in some way, e.g., in an attribute recording the time of observation. Thus, two *identical* records should really merge into the same record.

3. GENERIC ENTITY RESOLUTION

Given the match and merge functions, we can now ask what is the correct result of an entity resolution algorithm. It is clear that if two records match, they should be merged together. If the merged record matches another record, then those two should be merged together as well. But what should happen to the original matching records? Consider:

$r_1 = 0.8[name : \text{Alice}, areacode : 202]$

$r_2 = 0.7[name : \text{Alice}, phone : \text{555-1212}]$.

The merge of the two records might be:

$r_{12} = 0.56[name : \text{Alice},\ areacode : 202,\ phone : \text{555-1212}]$

In this case, the merged record has all of the information in r_1 and r_2, but with a lower confidence. So dropping the original two records would lose information. Therefore, to be conservative, the result of an entity resolution algorithm must contain the original records as well as records derived through merges. Based on this intuition, we define the correct result of entity resolution as follows.

Definition 3.1. *Given a set of records R, the* result of Entity Resolution *$ER(R)$ is the smallest set S such that:*

1. *$R \subseteq S$,*
2. *For any records $r_1, r_2 \in S$, if $r_1 \approx r_2$, then $\langle r_1, r_2 \rangle \in S$.*

We say that S_1 is smaller than S_2 if $S_1 \subseteq S_2$. The terminology "smallest" implies that there exists a unique result, which is proven in the extended version of this paper [15].

Intuitively, $ER(R)$ is the set of all records that can be derived from the records in R, or from records derived from them. A natural "brute-force" algorithm (BFA) for computing $ER(R)$ involves comparing all pairs, merging those that match, and repeating until no new records are found. This algorithm is presented formally in the extended version of this paper [15].

4. KOOSH

A brute-force algorithm like BFA is inefficient, essentially because the results of match comparisons are forgotten after every iteration. As an example, suppose $R = r_1, r_2$, $r_1 \approx r_2$, and $\langle r_1, r_2 \rangle$ doesn't match anything. In the first round, BFA will compare r_1 with r_2, and merge them together, adding $\langle r_1, r_2 \rangle$ to the set. In the second round, r_1 will be compared with r_2 a second time, and then merged together again. This comparison is redundant. In data sets with more records, the number of redundant comparisons is even greater.

We give in Figure 1 the Koosh algorithm, which improves upon BFA by removing these redundant comparisons. The algorithm works by maintaining two sets. R is the set of records that have not been compared yet, and R' is a set of records that have all been compared with each other. The algorithm works by iteratively taking a record r out of R, comparing it to every record in R', and then adding it to R'. For each record r' that matched r, the record $\langle r, r' \rangle$ will be added to R.

Using our simple example, we illustrate the fact that redundant comparisons are eliminated. Initially, $R = \{r_1, r_2\}$ and $R' = \emptyset$. In the first iteration, r_1 is removed from R and compared against everything in R'. There is nothing in R', so there are no matches, and r_1 is added to R'. In the second iteration, r_2 is removed and compared with everything in R', which consists of r_1. Since $r_1 \approx r_2$, the two records are merged and $\langle r_1, r_2 \rangle$ is added to R. Record r_2 is added to R'. In the third iteration, $\langle r_1, r_2 \rangle$ is removed from R and compared against r_1 and r_2 in R'. Neither matches, so $\langle r_1, r_2 \rangle$ is added to R'. This leaves R empty, and the algorithm terminates. In the above example, r_1 and r_2 were compared against each other only once, so the redundant comparison that occurred in BFA has been eliminated.

The Koosh algorithm correctly computes $ER(R)$. Moreover, it is efficient. No other algorithm that computes $ER(R)$ can perform fewer comparisons. These facts are proven in the extended version of this paper.

Theorem 4.6. Koosh is optimal, in the sense that no algorithm that computes $ER(R)$ makes fewer comparisons.

input: a set R of records
output: a set R' of records, $R' = ER(R)$
$R' \leftarrow \emptyset$
while $R \neq \emptyset$ **do**
  $r \leftarrow$ a record from R
  remove r from R
  for all $r' \in R'$ **do**
    if $r \approx r'$ **then**
      merged $\leftarrow \langle r, r' \rangle$
      if *merged* $\notin R \cup R' \cup \{r\}$ **then**
        add *merged* to R
      end if
    end if
  end for
  add r to R'
end while
return R'

Algorithm 1: The Koosh algorithm for ER(R)

5. DOMINATION

Even though Koosh is quite efficient, it is still very expensive, especially since the answer it must compute can be very large. In this section and the next two, we explore ways to tame this complexity, by exploiting additional properties of the match and merge functions (Section 6), or by only computing a still-interesting subset of the answer (using thresholds, in Section 7, or the notion of domination, which we introduce next).

To motivate the concept of domination, consider the following records r_1 and r_2, that match, and merge into r_3:
$r_1 = 0.8[name : \text{Alice},\ areacode : 202]$
$r_2 = 0.7[name : \text{Alice},\ phone : 555\text{-}1212]$.
$r_3 = 0.7[name : \text{Alice},\ areacode : 202,\ phone : 555\text{-}1212]$.

The resulting r_3 contains all of the attributes of r_2, and its confidence is the same. In this case it is natural to consider a "dominated" record like r_2 to be redundant and unnecessary. Thus, a user may only want the ER answer to contain only non-dominated records. These notions are formalized by the following definitions.

Definition 5.1. *We say that a record r* dominates *a record s, denoted $s \leq r$, if the following two conditions hold:*

1. *$s.\mathcal{A} \subseteq r.\mathcal{A}$*
2. *$s.\mathcal{C} \leq r.\mathcal{C}$*

Definition 5.2. *Given a set of base records R, the non-dominated entity-resolved set, $NER(R)$ contains all records in $ER(R)$ that are* non-dominated. *That is, $r \in NER(R)$ if and only if $r \in ER(R)$ and there does not exist any $s \in ER(R)$, $s \neq r$, such that $r \leq s$.*

Note that just like $ER(R)$, $NER(R)$ may be infinite. In the case that $ER(R)$ is finite, one way to compute $NER(R)$ is to first compute $ER(R)$ and then remove dominated records. This strategy does not save much effort since we still have to compute $ER(R)$. A significant performance improvement is to discard a dominated record as soon as it is found in the resolution process, on the premise that a dominated record will never participate in the generation of a non-dominated record. This premise is stated formally as follows:

- *Domination Property:* If $s \leq r$ and $s \approx x$ then $r \approx x$ and $\langle s, x \rangle \leq \langle r, x \rangle$.

This domination property may or may not hold in a given application. For instance, let us return to our r_1, r_2, r_3 example at the beginning of this section. Consider a fourth record $r_4 = 0.9[name : \text{Alice}, areacode : 717, phone : 555\text{-}1212, age : 20]$. A particular match function may decide that r_4 does *not* match r_3 because the area codes are different, but r_4 and r_2 may match since this conflict does not exist with r_2. In this scenario, we cannot discard r_2 when we generate a record that dominates it (r_3), since r_2 can still play a role in some matches.

However, in applications where having more information in a record can never reduce its match chances, the domination property can hold and we can take advantage of it. If the domination property holds then we can throw away dominated records as we find them while computing $NER(R)$. We prove this fact in the extended version of this paper.

5.1 Algorithm Koosh-ND

Koosh can be modified to eliminate dominated records early as follows. First, Koosh-ND begins by removing all dominated records from the input set. Second, within the body of the algorithm, whenever a new merged record m is created (line 10), the algorithm checks whether m is dominated by any record in R or R'. If so, then m is immediately discarded, before it is used for any unnecessary comparisons. Note that we do *not* check if m dominates any other records, as this check would be expensive in the inner loop of the algorithm. Finally, since we do not incrementally check if m dominates other records, we add a step at the end to remove all dominated records from the output set.

Koosh-ND relies on two complex operations: removing all dominated records from a set and checking if a record is dominated by a member of a set. These seem like expensive operations that might outweigh the gains obtained by eliminating the comparisons of dominated records. However, using an inverted list index that maps label-value pairs to the records that contain them, we can make these operations quite efficient.

The correctness of Koosh-ND is proven in the extended version of this paper.

6. THE PACKAGES ALGORITHM

In Section 3, we illustrated why ER with confidences is expensive, on the records r_1 and r_2 that merged into r_3:

$r_1 = 0.8[name : \text{Alice}, areacode : 202]$,

$r_2 = 0.7[name : \text{Alice}, phone : 555\text{-}1212]$,

$r_3 = 0.56[name : \text{Alice}, areacode : 202, phone : 555\text{-}1212]$.

Recall that r_2 cannot be discarded essentially because it has a higher confidence than the resulting record r_3. However, notice that other than the confidence, r_3 contains more label-value pairs, and hence, if it were not for its higher confidence, r_2 would not be necessary. This observation leads us to consider a scenario where the records minus confidences can be resolved efficiently, and then to add the confidence computations in a second phase.

In particular, let us assume that our merge function is "information preserving" in the following sense: When a record r merges with other records, the information carried by r's attributes is not lost. We formalize this notion of "information" by defining a relation "$\sqsubseteq$": $r \sqsubseteq s$ means that the attributes of s carry more information than those of r. We assume that this relation is transitive. Note that $r \sqsubseteq s$ and $s \sqsubseteq r$ does *not* imply that $r = s$; it only implies that $r.\mathcal{A}$ carries as much information as $s.\mathcal{A}$.

The property that merges are information preserving is formalized as follows:

- *Property P1:* If $r \approx s$ then $r \sqsubseteq \langle r, s \rangle$ and $s \sqsubseteq \langle r, s \rangle$.
- *Property P2:* If $s \sqsubseteq r$, $s \approx x$ and $r \approx x$, then $\langle s, x \rangle \sqsubseteq \langle r, x \rangle$

For example, a merge function that unions the attributes of records would have properties P1 and P2. Such functions are common in "intelligence gathering" applications, where one wishes to collect all information known about entities, even if contradictory. For instance, say two records report different passport numbers or different ages for a person. If the records merge (e.g., due to evidence in other attributes) such applications typically gather all the facts, since the person may be using fake passports reporting different ages.

Furthermore, we assume that adding information to a record does not change the outcome of match. In addition, we also assume that the match function does not consider confidences, only the attributes of records. These characteristics are formalized by:

- *Property P3:* If $s \sqsubseteq r$ and $s \approx x$, then $r \approx x$.

Having a match function that ignores confidences is not very constraining: If two records are unlikely to match due to low confidences, the merge function can still assign a low confidence to the resulting record to indicate it is unlikely. The second aspect of Property P3 rules out "negative evidence": adding information to a record cannot rule out a future match. However, negative information can still be handled by decreasing the confidence of the resulting record.

The algorithm of Figure 2 exploits these properties to perform ER more efficiently. It proceeds in two phases: a first phase bypasses confidences and groups records into disjoint packages. Because of the properties, this first phase can be done efficiently, and records that fall into different packages are known not to match. The second phase runs ER with confidences on each package separately. We next explain and justify each of these two phases.

6.1 Phase 1

In Phase 1, we may use any generic ER algorithm, such as those in [2] to resolve the base records, but with some additional bookkeeping. For example, when two base records r_1 and r_2 merge into r_3, we combine all three records together into a *package* p_3. The package p_3 contains two things: (i) a root $r(p_3)$ which in this case is r_3, and (ii) the base records $b(p_3) = \{r_1, r_2\}$.

Actually, base records can also be viewed as packages. For example, record r_2 can be treated as package p_2 with $r(p_2) = r_2$, $b(p_2) = \{r_2\}$. Thus, the algorithm starts with a set of packages, and we generalize our match and merge functions to operate on packages.

For instance, suppose we want to compare p_3 with a package p_4 containing only base record r_4. That is, $r(p_4) = r_4$ and $b(p_4) = \{r_4\}$. To compare the packages, we only compare their roots: That is, $M(p_3, p_4)$ is equivalent to $M(r(p_3), r(p_4))$, or in this example equivalent to $M(r_3, r_4)$. (We use the same symbol M for record and package matching.) Say these records do match, so we generate a new

input: a set R of records
output: a set R' of records, $R' = ER(R)$
Define for Packages:
match: $p \approx p'$ iff $r(p) \approx r(p')$
merge: $\langle p, p' \rangle = p''$:
    with root: $r(p'') = \langle r(p), r(p') \rangle$
    and base: $b(p'') = b(p) \cup b(p')$
Phase 1:
$P \leftarrow \emptyset$
for all records rec in R **do**
  create package p:
    with root: $r(p) = rec$
    and base: $b(p) = \{rec\}$
  add p to P
end for
compute $P' = ER(P)$ (e.g., using Koosh) with the following modification: Whenever packages p, p' are merged into p'', delete p and p' immediately, then proceed.
Phase 2:
$R' \leftarrow \emptyset$
for all packages $p \in P'$ **do**
  compute $Q = ER(b(p))$ (e.g. using Koosh)
  add all records in Q to R'
end for
return R'

Algorithm 2: The Packages algorithm

package p_5 with $r(p_5) = \langle r_3, r_4 \rangle$ and $b(p_5) = b(p_3) \cup b(p_4) = \{r_1, r_2, r_4\}$.

The package p_5 represents not only the records in $b(p_5)$, but also any records that can be derived from them. That is, p_5 represents all records in $ER(b(p_5))$. For example, p_5 implicitly represents the record $\langle r_1, r_4 \rangle$, which may have a higher confidence that the root of p_5. Let us refer to the complete set of records represented by p_5 as $c(p_5)$, i.e., $c(p_5) = ER(b(p_5))$. Note that the package does not contain $c(p_5)$ explicitly, the set is just implied by the package.

The key property of a package p is that the attributes of its root $r(p)$ carry more information (or the same) than the attributes of any record in $c(p)$, that is for any $s \in c(p)$, $s \sqsubseteq r(p)$. This property implies that any record u that does *not* match $r(p)$, cannot match any record in $c(p)$.

THEOREM 6.3. For any package p, if a record u does not match the root $r(p)$, then u does not match any record in $c(p)$.

This fact in turn saves us a lot of work! In our example, once we wrap up base records r_1, r_2 and r_4 into p_5, we do not have to involve them in any more comparisons. We only use $r(p_5)$ for comparing against other packages. If p_5 matches some other package p_8 (i.e., the roots match), we merge the packages. Otherwise, p_5 and p_8 remain separate since they have nothing in common. That is, nothing in $c(p_5)$ matches anything in $c(p_8)$.

6.2 Phase 2

At the end of Phase 1, we have resolved all the base records into a set of independent packages. In Phase 2 we resolve the records in each package, now taking into account confidences. That is, for each package p we compute $ER(b(p))$, using an algorithm like Koosh. Since none of the records from other packages can match a record in $c(p)$, the $ER(b(p))$ computation is completely independent from the other computations. Thus, we save a very large number of comparisons in this phase where we must consider the different order in which records can merge to compute their confidences. The more packages that result from Phase 1, the finer we have partitioned the problem, and the more efficient Phase 2 will be.

6.3 Packages-ND

As with Koosh, there is a variant of Packages that handles domination. To remove dominated records from the final result, we simply use Koosh-ND in Phase 2 of the Packages algorithm. Note that it is not necessary to explicitly remove dominated packages in Phase 1. To see this, say at some point in Phase 1 we have two packages, p_1 and p_2 such that $r(p_1) \leq r(p_2)$, and hence $r(p_1) \sqsubseteq r(p_2)$. Then p_1 will match p_2 (by Property P3 and idempotence), and both packages will be merged into a single one, containing the base records of both.

7. THRESHOLDS

Another opportunity to reduce the resolution workload lies within the confidences themselves. Some applications may not need to know every record that could possibly be derived from the input set. Instead, they may only care about the derived records that are above a certain confidence threshold.

DEFINITION 7.1. *Given a threshold value T and a set of base records R, we define the above-threshold entity-resolved set, $TER(R)$ that contains all records in $ER(R)$ with confidences above T. That is, $r \in TER(R)$ if and only if $r \in ER(R)$ and $r.\mathcal{C} \geq T$.*

As we did with domination, we would like to remove below-threshold records, not after completing the resolution process (as suggested by the definition), but as soon as they appear. However, we will only be able to remove below-threshold records if they cannot be used to derive above-threshold records. Whether we can do that depends on the semantics of confidences.

As we mentioned earlier, models for the interpretation of confidences vary. Under some interpretations, two records with overlapping information might be considered as independent evidence of a fact, and the merged record will have a higher confidence than either of the two base records.

Other interpretations might see two records, each with their own uncertainty, and a match and merge process which is also uncertain, and conclude that the result of a merge must have lower confidence than either of the base records. For example, one interpretation of $r.\mathcal{C}$ could be that it is the probability that r correctly describes a real-world entity. Using the "possible worlds" metaphor [13], if there are N equally-likely possible worlds, then an entity containing at least the attributes of r will exist in $r.\mathcal{C} \times N$ worlds. With this interpretation, if r_1 correctly describes an entity with probability 0.7, and r_2 describes an entity with probability 0.5, then $\langle r_1, r_2 \rangle$ cannot be true in more worlds than r_2, so its confidence would have to be less than or equal to 0.5.

To be more formal, some interpretations, such as the example above, will have the following property.

- *Threshold Property:* If $r \approx s$ then $\langle r, s \rangle.\mathcal{C} \leq r.\mathcal{C}$ and $\langle r, s \rangle.\mathcal{C} \leq s.\mathcal{C}$.

Given the threshold property, we can compute $TER(R)$ more efficiently. In the extended version of this paper, we prove that if the threshold property holds, then all results can be obtained from above-threshold records.

7.1 Algorithms Koosh-T and Koosh-TND

As with removing dominated records, Koosh can be easily modified to drop below-threshold records. First, we add an initial scan to remove all base records that are already below threshold. Then, we simply add the following conjunct to the condition of Line 10 of the algorithm:

$$merged.\mathcal{C} \geq T$$

Thus, merged records are dropped if they are below the confidence threshold.

THEOREM 7.2. When $TER(R)$ is finite, Koosh-T terminates and computes $TER(R)$.

By performing the same modification as above on Koosh-ND, we obtain the algorithm Koosh-TND, which computes the set $NER(R) \cap TER(R)$ of records in $ER(R)$ that are neither dominated nor below threshold.

7.2 Packages-T and Packages-TND

If the threshold property holds, Koosh-T or Koosh-TND can be used for Phase 2 of the Packages algorithm, to obtain algorithm Packages-T or Packages-TND. In that case, below-threshold and/or dominated records are dropped as each package is expanded.

8. EXPERIMENTS

To summarize, we have discussed three main algorithms: BFA, Koosh, and Packages. For each of those basic three, there are three variants, adding in thresholds (T), non-domination (ND), or both (TND). In this section, we will compare the three algorithms against each other using both thresholds and non-domination. We will also investigate how performance is affected by varying threshold values, and, independently, by removing dominated records.

To test our algorithms, we ran them on synthetic data. Synthetic data gives us the flexibility to carefully control the distribution of confidences, the probability that two records match, as well as other important parameters. Our goal in generating the data was to emulate a realistic scenario where n records describe various aspects of m real-world entities ($n > m$). If two of our records refer to the same entity, we expect them to match with much higher probability than if they referred to different entities.

To emulate this scenario, we assume that the real-world entities can be represented as points on a number line. Records about a particular entity with value x contain an attribute A with a value "close" to x. (The value is normally distributed with mean x, see below.) Thus, the match function can simply compare the A attribute of records: if the values are close, the records match. Records are also assigned a confidence, as discussed below.

For our experiments we use an "intelligence gathering" merge function as discussed in Section 6, which unions attributes. Thus, as a record merges with others, it accumulates A values and increases its chances of matching other records related to the particular real-world entity.

To be more specific, our synthetic data was generated using the following parameters (and their default values):

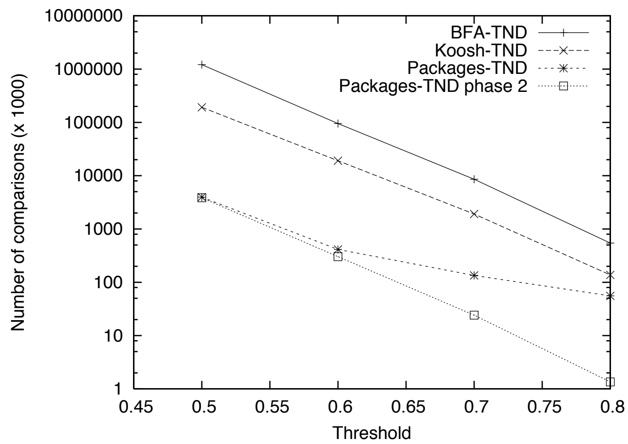

Figure 1: Thresholds vs. Matches

- n, the number of records to generate (default: 1000)
- m, the number of entities to simulate (default: 100)
- $margin$, the separation between entities (default: 75)
- σ, the standard deviation of the normal curve around each entity. (default: 10)
- μ_c, the mean of the confidence values (default: 0.8)

To generate one record r, we proceed as follows: First, pick a uniformly distributed random integer i in the range $[0, m-1]$. This integer represents the value for the real-word entity that r will represent. For the A value of r, generate a random floating point value v from a normal distribution with standard deviation σ and a mean of $margin \cdot i$. To generate r's confidence, compute a uniformly distributed value c in the range $[\mu_c - 0.1, \mu_c + 0.1]$ (with $\mu_c \in [0.1, 0.9]$ so that c stays in $[0, 1]$). Now create a record r with $r.\mathcal{C} = c$ and $r.\mathcal{A} = \{A : v\}$. Repeat all of these steps n times to create n synthetic records.

Our merge function takes in the two records r_1 and r_2, and creates a new record r_m, where $r_m.\mathcal{C} = r_1.\mathcal{C} \times r_2.\mathcal{C}$ and $r_m.\mathcal{A} = r_1.\mathcal{A} \cup r_2.\mathcal{A}$. The match function detects a match if for the A attribute, there exists a value v_1 in $r_1.\mathcal{A}$ and a value v_2 in $r_2.\mathcal{A}$ where $|v_1 - v_2| < k$, for a parameter k chosen in advance ($k = 25$ except where otherwise noted).

Naturally, our first experiment compares the performance of our three algorithms, BFA-TND, Koosh-TND and Packages-TND, against each other. We varied the threshold values to get a sense of how much faster the algorithms are when a higher threshold causes more records to be discarded. Each algorithm was run at the given threshold value three times, and the resulting number of comparisons was averaged over the three runs to get our final results.

Figure 1 shows the results of this first experiment. The first three lines on the graph represent the performance of our three algorithms. On the horizontal axis, we vary the threshold value. The vertical axis (logarithmic) indicates the number of calls to the match function, which we use as a measure of the work performed by the algorithms. The first thing we notice is that work performed by the algorithms grows exponentially as the threshold is decreased. Thus, clearly thresholds are a very powerful tool: one can get high-

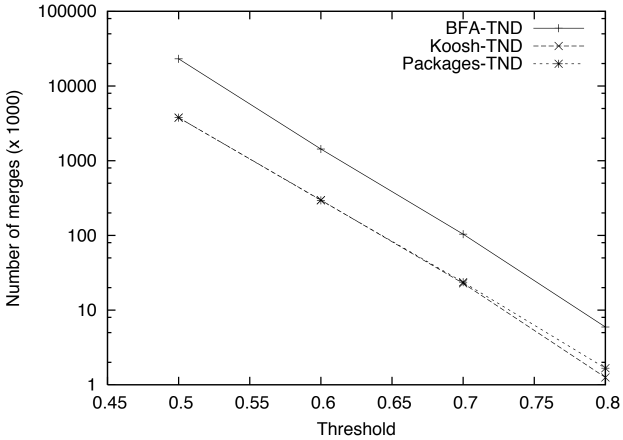

Figure 2: Thresholds vs. Merges

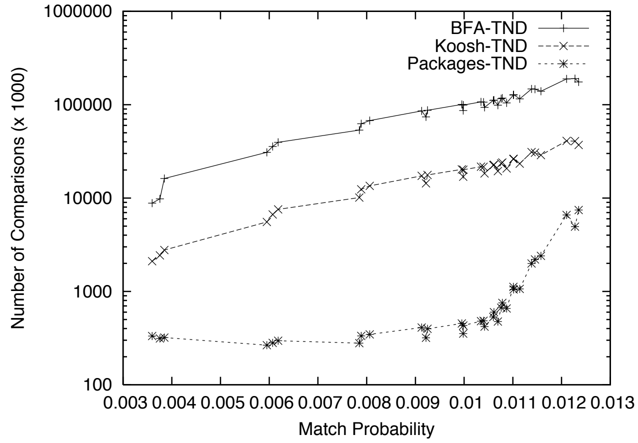

Figure 3: Selectivity vs. Comparisons

confidence results at a relatively modest cost, while computing the lower confidence records gets progressively more expensive! Also interestingly, the BFA-TND and Koosh-TND lines are parallel to each other. This means that they are consistently a constant factor apart. Roughly, BFA does 10 times the number of comparisons that Koosh does.

The Packages-TND algorithm is far more efficient than the other two algorithms. Of course, Packages can only be used if Properties P1, P2 and P3 hold, but when they do hold, the savings can be dramatic. We believe that these savings can be a strong incentive for the application expert to design match and merge function that satisfy the properties.

We also compared our algorithms based on the number of merges performed. In Figure 2, the vertical axis indicates the number of merges that are performed by the algorithms. We can see that Koosh-TND and the Packages-TND are still a great improvement over BFA. BFA performs extra merges because in each iteration of its main loop, it recompares all records and merges any matches found. The extra merges result in duplicate records which are eliminated when they are added to the result set. Packages performs slightly more merges than Koosh, since the second phase of the algorithm does not use any of the merges that occurred in the first phase. If we subtract the Phase 1 merges from Packages (not shown in the figure), Koosh and Packages perform roughly the same number of merges.

In our next experiment, we compare the performance of our algorithms as we vary the probability that base records match. We can control the match probability by changing parameters k or σ, but we use the resulting match probability as the horizontal axis to provide more intuition. In particular, to generate Figure 3, we vary parameter k from 5 to 55 in increments of 5 (keeping the threshold value constant at 0.6). During each run, we measure the match probability as the fraction of base record matches that are positive. (The results are similar when we compute the match probability over all matches.) For each run, we then plot the match probability versus the number of calls to the match function, for our three algorithms.

As expected, the work increases with greater match probability, since more records are produced. Furthermore, we note that the BFA and Koosh lines are roughly parallel, but the Packages line stays level until a quick rise in the amount of work performed once the match probability reaches about 0.011. The Packages optimization takes advantage of the fact that records can be separated into packages that do not merge with one another.

In practice, we would expect to operate in the range of Figure 3 where the match probability is low and Packages outperforms Koosh. In our scenario with high match probabilities, records that refer to different entities are being merged, which means the match function is not doing its job. One could also get high match probabilities if there were very few entities, so that packages do not partition the problem finely. But again, in practice one would expect records to cover a large number of entities.

9. RELATED WORK

Originally introduced by Newcombe et al. [17] under the name of record linkage, and formalized by Fellegi and Sunter [9], the ER problem was studied under a variety of names, such as Merge/Purge [12], deduplication [18], reference reconciliation [8], object identification [21], and others. Most of the work in this area (see [23, 11] for recent surveys) focuses on the "matching" problem, i.e., on deciding which records do represent the same entities and which ones do not. This is generally done in two phases: Computing measures of how similar atomic values are (e.g., using edit-distances [20], TF-IDF [6], or adaptive techniques such as q-grams [4]), then feeding these measures into a model (with parameters), which makes matching decisions for records. Proposed models include unsupervised clustering techniques [12, 5], Bayesian networks [22], decision trees, SVM's, conditional random fields [19]. The parameters of these models are learned either from a labeled training set (possibly with the help of a user, through active learning [18]), or using unsupervised techniques such as the EM algorithm [24].

All the techniques above manipulate and produce numerical values, when comparing atomic values (e.g. TF-IDF scores), as parameters of their internal model (e.g., thresholds, regression parameters, attribute weights), or as their output. But these numbers are often specific to the techniques at hand, and do not have a clear interpretation in terms of "confidence" in the records or the values. On the other hand, representations of uncertain data exist, which soundly model confidence in terms of probabilities (e.g., [1, 10]), or beliefs [14]. However these approaches focus on computing the results and confidences of exact queries, extended with simple "fuzzy" operators for value comparisons (e.g., see [7]), and are not capable of any advanced form of entity resolution. We propose a flexible solution for ER that accommodates any model for confidences, and proposes efficient algorithms based on their properties.

Our generic approach departs from existing techniques in that it interleaves merges with matches. The first phase of the Packages algorithm is similar to the set-union algorithm described in [16], but our use of a merge function allows the selection of a true representative record. The presence of "custom" merges is an important part of ER, and it makes confidences non-trivial to compute. The need for iterating matches and merges was identified by [3] and is also used in [8], but their record merges are simple aggregations (similar to our "information gathering" merge), and they do not consider the propagation of confidences through merges.

10. CONCLUSION

In this paper we look at ER with confidences as a "generic database" problem, where we are given black-boxes that compare and merge records, and we focus on efficient algorithms that reduce the number of calls to these boxes. The key to reducing work is to exploit generic properties (like the threshold property) than an application may have. If such properties hold we can use the optimizations we have studied (e.g., Koosh-T when the threshold property holds). Of the three optimizations, thresholds is the most flexible one, as it gives us a "knob" (the threshold) that one can control: For a high threshold, we only get high-confidence records, but we get them very efficiently. As we decrease the threshold, we start adding lower-confidence results to our answer, but the computational cost increases. The other two optimizations, domination and packages, can also reduce the cost of ER very substantially but do not provide such a control knob.

11. REFERENCES

[1] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, 1992.

[2] O. Benjelloun, H. Garcia-Molina, J. Jonas, Q. Su, and J. Widom. Swoosh: A generic approach to entity resolution. Technical report, Stanford University, 2005.

[3] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *Proc. of the SIGMOD 2004 Workshop on Research Issues on Data Mining and Knowledge Discovery*, June 2004.

[4] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proc. of ACM SIGMOD*, pages 313–324. ACM Press, 2003.

[5] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *Proc. of ICDE*, Tokyo, Japan, 2005.

[6] William Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18:288–321, 2000.

[7] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[8] X. Dong, A. Y. Halevy, J. Madhavan, and E. Nemes. Reference reconciliation in complex information spaces. In *Proc. of ACM SIGMOD*, 2005.

[9] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[10] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[11] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: Current practice and future directions. Technical Report 03/83, CSIRO Mathematical and Information Sciences, 2003.

[12] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proc. of ACM SIGMOD*, pages 127–138, 1995.

[13] Saul Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[14] Suk Kyoon Lee. An extended relational database model for uncertain and imprecise information. In Li-Yan Yuan, editor, *VLDB*, pages 211–220. Morgan Kaufmann, 1992.

[15] David Menestrina, Omar Benjelloun, and Hector Garcia-Molina. Generic entity resolution with data confidences (extended version). Technical report, Stanford University, 2006.

[16] A. E. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *DMKD*, pages 0–, 1997.

[17] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130(3381):954–959, 1959.

[18] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of ACM SIGKDD*, Edmonton, Alberta, 2002.

[19] Parag Singla and Pedro Domingos. Object identification with attribute-mediated dependences. In *Proc. of PKDD*, pages 297 – 308, 2005.

[20] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[21] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems Journal*, 26(8):635–656, 2001.

[22] Vassilios S. Verykios, George V. Moustakides, and Mohamed G. Elfeky. A bayesian decision model for cost optimal record matching. *The VLDB Journal*, 12(1):28–40, 2003.

[23] W. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, Washington, DC, 1999.

[24] W. E. Winkler. Using the EM algorithm for weight computation in the fellegi-sunter model of record linkage. *American Statistical Association, Proceedings of the Section on Survey Research Methods*, pages 667–671, 1988.