Generic Entity Resolution in the SERF Project

Omar Benjelloun Hector Garcia-Molina Hideki Kawai Tait Eliott Larson
David Menestrina Qi Su Sutthipong Thavisomboon Jennifer Widom

Stanford University

Abstract

The SERF project at Stanford deals with the Entity Resolution (ER) problem, in which records determined to represent the same real-life "entities" (such as people or products) are successively located and combined. The approach we pursue is "generic", in the sense that the specific functions used to match and merge records are viewed as black boxes, which permits efficient, expressive and extensible ER solutions. This paper motivates and introduces the principles of generic ER, and gives an overview of the research directions we have been exploring in the SERF project over the past two years.

1 Introduction

Entity Resolution (ER) (also referred to as deduplication) is the process of identifying and merging records judged to represent the same real-world entity. ER is a well-known problem that arises in many applications. For example, mailing lists may contain multiple entries representing the same physical address, but each record may be slightly different, e.g., containing different spellings or missing some information. As a second example, consider a comparative shopping website, aggregating product catalogs from multiple merchants. Identifying records that *match*, i.e., records that represent the same product is challenging because there are no unique identifiers across merchant catalogs. A given product may appear in different ways in each catalog, and there is a fair amount of guesswork in determining which records match. Deciding if records match is often computationally *expensive*, e.g., may involve finding maximal common subsequences in two strings. How to *merge* records, i.e., combine records that match is often also *application dependent*. For example, say different prices appear in two records to be merged. In some cases we may wish to keep both of them, while in others we may want to pick just one as the "consolidated" price.

In the SERF project, we study ER as a "generic database problem". We say we take a generic approach because we do not study the internal details of the functions used to compare and merge records. Rather, we view these functions as "black-boxes" to be invoked by the ER engine. Given such black-boxes, we study algorithms for efficiently performing ER, i.e., we develop strategies that minimize the number of invocations to these potentially expensive black-boxes. An important component of our work is that we identify a set of properties that the black-boxes should have in order to lead to a well-defined single "answer" to the ER problem, as well as to efficient algorithms. For example, associative merges is one such important property: If merges are

Copyright 0000 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

not associative, the order in which records are merged may impact the final result. Our general framework for ER is introduced in Section 2, as well as our main algorithm.

Except for a few works such as [HS95], most existing work on ER focuses on developing techniques to achieve the best quality for ER, measured in terms of precision and recall, on some class of data or applications. In our generic approach, such metrics are dependent on the black-box functions, and our focus is rather on the framework and algorithms in which these black-boxes are used. Of course, to define a simple coherent setting for ER, we have to make some assumptions (e.g., that matches are computed pairwise), hence we only capture a subset of the (diverse) set of existing techniques for ER, a subset which we believe is useful for a large number of applications. We refer the reader to [BGMJ$^+$05] for a detailed review of related works.

Since ER is computationally expensive, we also developed strategies to distribute its computation across multiple processors. Again, we made our distributed ER algorithm generic, by providing simple abstractions that make the distribution strategy configurable, i.e., capable to accommodate the characteristics of data in specific applications. In particular, our abstractions provide a single unified way to express and leverage common forms of domain knowledge in order to "block" unnecessary record comparisons. For instance, if it is known that records representing products can only match if their prices are close enough, records can be split among processors based on their price, in a way that greatly reduces the communication costs, while still computing the correct result. Our distributed ER algorithm is presented in Section 3.

Because ER is an approximate process, it is often desirable to attach confidence values to the records, and propagate these confidences as matches and merges are performed. However, the meaning of confidences and the way they are propagated may vary. In some applications, the confidence of a record could be interpreted as the probability that it correctly represents an entity, while in others, confidences may measure the precision of data. We extend our model with confidences in a generic way, by leaving to the match and merge functions the responsibility to interpret and propagate confidences. As we illustrate in Section 4, adding confidences implies that some of the properties previously identified for the black-box functions do not hold anymore. For instance, the associativity of merges is often not satisfied, because the order in which records are matched and merged may quite naturally affect the confidence of a derived record. As a consequence, more expensive algorithms are needed for ER with confidences. However, some optimizations may reduce the cost of the ER computation, e.g., when the properties still apply for the data component of the records, or if only records with confidence above some threshold are of interest.

We conclude this paper with a discussion of current and future research directions in Section 5.

2 Generic Entity Resolution

We start by defining our generic model for ER. The input of ER is a set of *records*, and so is its output. We do not make any assumption about the particular form or data model used to represent records.

As an example, in Figure 1 we consider records representing products, along the lines of the comparison shopping scenario mentioned in the introduction. Each product has a name, a price (or price range), and a category. This example is inspired from the data used in our experiments, which consists of actual product descriptions provided to us by the Yahoo! Shopping team.

	name	price	category
r_1	Apple iPod	249	MP3 player
r_2	Apple iPod	299	
r_3	iPod	270	MP3 player

Figure 1: Product records

2.1 Match and Merge

Our generic ER model is based on two black-box functions provided as input to the ER computation: match and merge.

A *match function* M is a function that takes two records as input and returns a Boolean value. Function M returns true if the input records represent the same entity, and false otherwise. Such a match function reflects the restrictions we are making that (i) matching decisions can be made “locally”, based on the two records being compared, and (ii) that such decisions are Boolean, and not associated with any kind of numeric confidence (we will revisit this restriction in Section 4). In practice, such functions are easier to write than functions that consider multiple records.

A *merge function* μ is a function that takes in two records and returns a single record. Function μ is only defined for pairs of matching records, i.e., records known to represent the same entity. Its output is a “consolidated” record representing that entity. If a record r is (transitively) obtained through a merge involving a record r', we say that r is *derived* from r'.

When M and μ are understood from the context, $M(r_1, r_2) = \texttt{true}$ (resp. $M(r_1, r_2) = \texttt{false}$) is denoted by $r_1 \approx r_2$ (resp. $r_1 \not\approx r_2$), and $\mu(r_1, r_2)$ is denoted by $\langle r_1, r_2 \rangle$.

To illustrate, we define sample match and merge functions for our comparison shopping example. The match function is based on product names being equal, or all attributes of the records being highly similar. Such a match function can be expressed as:

$$M(r1, r2) = (r_1.name == r_2.name) \vee (M_{name}(r_1, r_2) \wedge M_{price}(r_1, r_2) \wedge M_{category}(r_1, r_2))$$

Let us say that M_{name} computes some similarity n of record names (e.g., an edit-distance) and returns true if $n > 0.8$. M_{price} computes the relative distance p among prices, and returns true if $p > 0.9$. $M_{category}$ returns true if the two records have the exact same category.

With this match function, in Figure 1 $r_1 \approx r_2$, because they have the exact same name. However, $r_1 \not\approx r_3$ because their prices are too far apart, and $r_2 \not\approx r_3$ because r_2 does not have a category value.

For the merge function, let us assume that it has some way to normalize product names into a single name (e.g., by relying on an external source to find the closest reference product name), that it keeps a range for prices, and keeps the union of category values from the base records. This merge function would produce $r_4 = \langle r_1, r_2 \rangle$:

$r4$ = (Apple iPod, [249-299], MP3 Player)

Observe that, unlike r_1 and r_2, the obtained record r_4 may match r_3 because it has combined price and category information from r_1 and r_2. This example illustrates one of the difficulties of ER: it is not sufficient to compare base records to each other. Derived records must be recursively compared to the other records in the dataset.

2.2 Domination

A last notion we need to introduce before defining generic ER is domination. Intuitively, if two records r_1 and r_2 are about the same entity but r_1 holds more information than r_2, then r_2 is useless for representing this entity. We say that r_1 *dominates* r_2. In general, any partial order on records could be used to define domination. For each application, a different notion of domination may be suitable.

To capture the fact that domination is application specific, we rely on the match and the merge functions to define it: We say that r_1 dominates r_2 if $r_1 \approx r_2$ (i.e., the two records match), and $\langle r_1, r_2 \rangle = r_1$. The consistency conditions we will introduce shortly in Section 2.4 ensure that domination is a partial order on records. In our example, the reader can verify that r_4 dominates r_1 and r_2.

2.3 Generic ER

We are now equipped to define entity resolution: Given a set of input records R, an *Entity Resolution* of R, denoted $ER(R)$ is a set of records such that:

- Any record in $ER(R)$ is derived (through merges) from records in R;
- Any record that can be derived from R is either in $ER(R)$, or is dominated by a record in $ER(R)$;
- No two records in $ER(R)$ match, and no record in $ER(R)$ is dominated by any other.

2.4 Consistent ER

If match and merge are arbitrary functions, entity resolution of a set of records R may not exist, may not be unique, and may be infinite. In [BGMJ$^+$05], we introduce simple and practical conditions on the match and merge functions, which guarantee that ER is "consistent", i.e., that it exists, is unique and finite. These properties are the following:

- *Commutativity:* For any pair of records r_1, r_2, $r_1 \approx r_2$ is the same as $r_2 \approx r_1$, and if r_1 and r_2 match, then $\langle r_1, r_2 \rangle$ and $\langle r_2, r_1 \rangle$ produce the same record.
- *Reflexivity/Idempotence:* Any record r matches itself, and $\langle r, r \rangle = r$
- *Representativity:* If $r_3 = \langle r_1, r_2 \rangle$, then r_3 "represents" r_1 and r_2, in the sense that r_3 matches any record that matches r_1 or r_2.
- *Merge associativity:* For any records r_1, r_2, r_3, if $\langle r_1, \langle r_2, r_3 \rangle \rangle$ and $\langle \langle r_1, r_2 \rangle, r_3 \rangle$ exist, then they are equal. Intuitively, this property means that if there exists multiple derivations involving the same set of records, then they should all produce the same result.

As discussed in [BGMJ$^+$05], if some of these properties do not hold, the entity resolution problem becomes much more expensive. For instance, without merge associativity we must consider all possible orders in which records may match and merge. Extending ER with confidences (see Section 4) leads to such a situation.

2.5 The R-Swoosh algorithm

When the four properties introduced above are satisfied, we develop an efficient ER algorithm, R-Swoosh, that reduces the number of invocations to the match and merge functions.

R-Swoosh relies on two sets: R, which initially contains all the input records, and R', which maintains the set of (so far) non-dominated, non-matching records. R-Swoosh successively compares each record in R to all the records present in R'. R-Swoosh performs a merge as soon as a pair of matching records is found. The obtained record is added to R, and the pair of matching records is deleted immediately. The algorithm terminates when R is empty, and R' contains $ER(R)$.

To illustrate, consider the run of R-Swoosh given in Figure 2. The algorithm starts with all the input records in R, and an empty R'. At every round, one record from R is compared to the records in R' and moved to R' if no match is found. Here, r_1 and r_2 are successively moved to R'. At round i, r_3 is compared to r_1 and a match is found. The two records are immediately merged into r_7, which is put back into R, while r_1 and r_3 are deleted. The algorithm ends when R is empty. R' contains the result of ER.

Intuitively, R-Swoosh is efficient because it interleaves matches, merges and deletions of dominated records. R-Swoosh may end up comparing all pairs of records to each other, but it eagerly performs merges and deletions as early as possible, thereby avoiding unnecessary future match comparisons. In [BGMJ$^+$05], it is shown that

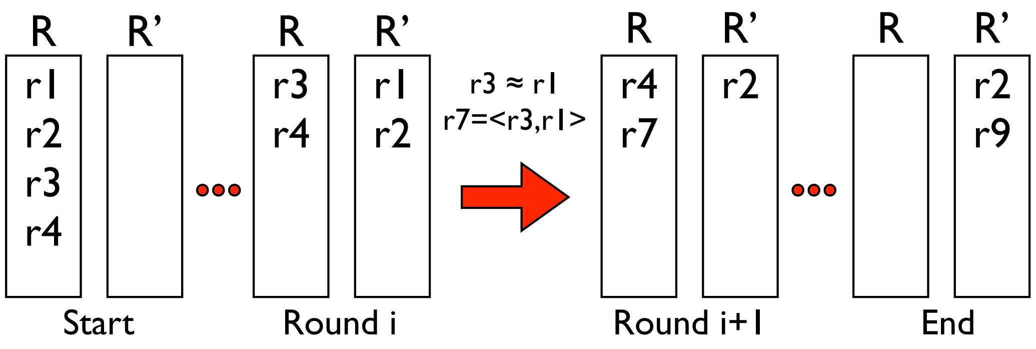

Figure 2: A sample run of the R-Swoosh algorithm

R-Swoosh is optimal, in the sense that no other algorithm can perform fewer record comparisons in the worst case.

In [BGMJ$^+$05], we also give a variant of the algorithm called F-Swoosh (F stands for “feature”) that efficiently caches the results of value comparisons when the match function can be expressed as a disjunction of match functions over “features”, i.e. parts of the records.

3 Distributed ER

Even though R-Swoosh is optimal, ER is still an expensive process, as it may need to perform (expensive) match comparisons on each pair of records in the (often large) input dataset. To deal with this complexity, we investigated in [BGMK$^+$06].ways to parallelize the ER computation across multiple processors.

Distributing data to processors for the ER computation requires care, as any pair of records is a potential match, and therefore needs to be compared. Also, recall that records produced through merges need to be compared with others, and must be distributed adequately.

In general, there is no optimal, application-independent strategy to distribute data to processors. Depending on the application and the distribution of values, some strategy may be more sensible than others. In particular, it is important to exploit any domain knowledge that saves some comparisons, by reflecting it in the distribution strategy.

As an example, in our comparison shopping application, we may have the domain knowledge that prices for the same product never vary by more than, say, 20% from one vendor to another. We can exploit this knowledge by making each processor responsible for one price segment, with a 20% overlap to account for prices close to segment boundaries. However, prices may not be uniformly distributed, in which case we should distribute the workload of crowded segments across multiple processors.

To support such distribution needs in a generic way, we introduce two abstract functions:

- *scope:* captures the distribution of records to processors, by assigning to each record a set of processors,
- *resp:* determines which processors are responsible of comparing which pairs of records.

We use these functions as primitives in our distributed ER algorithm. To guarantee the correctness of the algorithm, the scope and resp functions need only satisfy a simple *coverage* property: any pair of potentially matching records have scopes that intersect at least at one processor, which is responsible for comparing them.

The D-Swoosh algorithm runs a variant of R-Swoosh at each of the processors. Initially, records are distributed to processors based on the scope function. Then, each processor operates in a similar fashion to R-Swoosh, with the main difference that processors asynchronously exchange messages about which records are added or deleted (again, using the scope function), and keep track of all the records they know have been deleted. The algorithm terminates when all processors are idle and no more messages are exchanged. The result of ER is the union of the R' sets at each processor.

Figure 3 illustrates part of the computation at one of the processors. P_i discovers that records r_3 and r_1 match, and merges them into r_7. It sends an $add(r_7)$ message to P_{27} and P_{67}, the processors in $scope(r_7)$, and delete messages for r_1 and r_3 to the processors in their respective scopes (including itself). Upon receiving these add and delete messages, the processors update their local R and R' sets accordingly. The full D-Swoosh algorithm is described in [BGMK$^+$06].

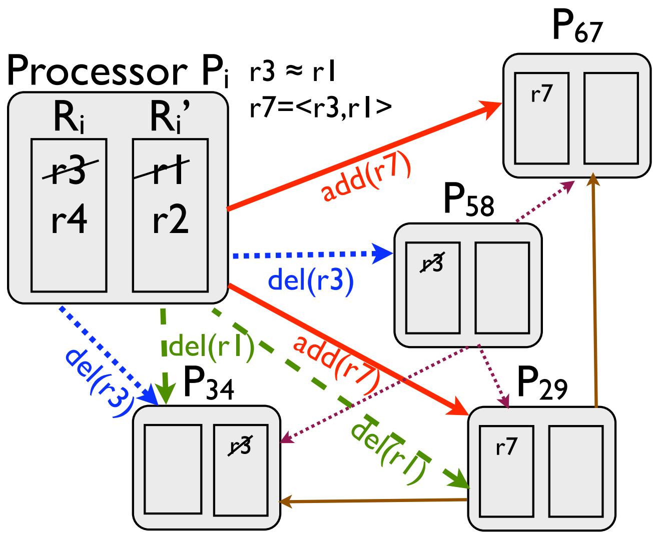

Figure 3: The D-Swoosh algorithm

In [BGMK$^+$06], we also provide a “library” of scope and resp functions, both for the case when no domain knowledge is available, and all pairs of records must be compared, and for the case where domain knowledge of some common forms is available. In the latter case, we are able to express and leverage necessary conditions on pairs of matching records such as the equality on the value of some attribute (such as the category in our example), a linearly ordered attribute with a sliding window (such as our price example), or an ancestor/descendant relationship in a hierarchy. We experimentally compared the different schemes (both with and without domain knowledge) on a comparison shopping dataset from Yahoo!.

4 ER with confidences

In the model we presented so far, everything is certain: records are exact facts, and the match function makes Boolean decision on whether pairs of records represent the same entity. The properties of the match and merge function essentially guarantee that there exists a unique ER solution. However, manipulating numerical confidences (or uncertainties) is often desirable in ER. For instance, input records may come from unreliable sources, and may have confidences associated with them. The match comparisons between records may also yield confidences that represent how likely the records are to represent the same real-world entity. Similarly, the merge process may introduce additional uncertainties, as there may not be a deterministic way to combine the information from different records. In each application domain, the interpretation of 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.

	name	price	category	*conf*
r_1	Apple iPod	249	MP3 player	0.8
r_2	Apple iPod	299		0.7
r_3	iPod	270	MP3 player	0.5

Figure 4: Product records with confidences

In [MBGM05], we extended our framework for ER with confidences in a generic way by simply associating a confidence value *conf* (between 0 and 1) to each record. The match and merge functions are responsible for manipulating and propagating confidence values. For instance, our product records may have initial confidences (e.g., reflecting the reliability of their sources), as shown in Figure 4.

The merge of r_1 and r_2 may now produce r_4, with a confidence value assigned by the merge function:

$r4$ = (Apple iPod, [249-299], MP3 Player) *conf:* 0.56

Here, the merge function multiplied the confidences of r_1 and r_2 to generate the confidence of r_4. This choice intuitively corresponds to making an independence assumption in a probabilistic interpretation of confidences. Other choices may also be reasonable, such as taking the minimum of the confidences of r_1 and r_2 as a confidence for r_4.

When confidences are present, the notion of domination must be extended. We say that r_1 dominates r_2 (with confidences) if the data component of r_1 dominates that of r_2 and the confidence of r_1 is higher than or equal to that of r_2. In our example, observe that r_4 does not dominate r_1 nor r_2, because it has a lower confidence.

Adding confidences significantly affects the ER process, because some of the properties introduced in Section 2.4 are not satisfied anymore:

- *No representativity:* The fact that two records r_1, r_2 match is inherently an uncertain operation, and therefore the record r_{12} produced by their merge is likely to have a lower confidence than both r_1 and r_2. Even if the data component of r_{12} "represents" that of r_1 and r_2, r_{12} may not match a record that r_1 or r_2 matches, because its has a lower confidence, an information that may be used by the match function. Therefore, the representativity property may not hold.

- *No Merge associativity:* The confidence of a derived record may depend on the sequence of merges that produced it. Given three records r_1, r_2, r_3, $\langle r_1, \langle r_2, r_3\rangle\rangle$ and $\langle\langle r_1, r_2\rangle, r_3\rangle$ may very well have different confidence values, e.g., because one of the derivation was based on "strong" match evidence (therefore yielding high confidence) while the other derivation follows a tenuous connection.

Because representativity and merge associativity may not hold, ER must be performed using a more expensive algorithm than R-Swoosh. Essentially, records participating in a merge cannot be deleted right away, and dominated records can only be removed after all possible matches and merges have been found. In [MBGM05], we provide *Koosh*, a variant of R-Swoosh which is the optimal sequential algorithm for generic ER when representativity and merge associativity do not hold. In a nutshell, Koosh also uses two sets R and R', but always compares records in R to *all* the records in R', and postpones the deletion of dominated records until R is empty.

For the important case where the properties of Section 2.4 do hold for the data component and are only violated because of confidences, we proposed a two-phase algorithm which performs much better than Koosh. The algorithm exploits the fact that matching on the data component is a necessary condition for matching in ER with confidences. It runs a first pass of R-Swoosh on the data component only to partition the base records into "packages", and then runs the more expensive Koosh algorithm on each package separately.

Another optimization we investigated was the use of thresholds to prune the search space: If the user is only interested in records with confidence above some fixed value, and if the meaning of confidences is such that they may only decrease upon merging records, then any record can be discarded as soon as its confidence falls below the threshold, because it cannot contribute to the derivation of any above-threshold record.

5 Conclusion

Entity resolution is a crucial information integration problem. We believe our generic approach provides a clean separation between the quality aspects of ER, encapsulated in the black-box match and merge functions, and its algorithmic and performance aspects, addressed by the sequential and distributed algorithms we developed. To conclude, we would like to mention some of the research directions we are currently investigating in the SERF project:

- *Large scale distributed ER:* We are in the process of deploying our D-Swoosh algorithm on a large-scale, shared-nothing distributed infrastructure consisting of tens of commodity PCs, to run ER on the full Yahoo! comparison shopping dataset (several Gigabytes). We hope to understand the performance and cost trade-offs involved in large scale distributed ER, and to develop optimization strategies to best adapt the distribution scheme (i.e., the scope and resp functions) to particular applications and datasets..
- *Negative ER and uncertainty:* Going beyond our essentially monotonic model for ER, we are incorporating negative information, to express constraints needed by a number of applications. Negative facts essentially lead to modeling uncertainty in the ER process, and embracing the fact that ER may have multiple alternative solutions, possibly with a probability distribution over them. We are building upon the ULDB model for databases with uncertainty and lineage [BSHW06]. Lineage, which keeps track of the derivation history of records is crucial to back-track previously made merge decisions, should new evidence suggest to do so. We are investigating efficient algorithms to compute the most probable ER answer in the presence of such negative information.
- *I/O's and buffer management for ER:* We are investigating strategies to efficiently perform ER when the dataset does not fit in main memory. We are developing and experimenting with various buffer management strategies, and corresponding adaptations of our ER algorithms.
- *Declarative ER:* We believe ER should be specified declaratively using match rules that combine atomic similarity functions on attribute values, and high level constraints able to capture applicable domain knowledge. Based on these specifications, which could be either entered by experts or learned from a training sample, we would like to derive an efficient "execution plan" for performing ER, possibly taking into account statistics on the atomic match and merge black-box "operators" through a suitable cost model.

References

[BGMJ$^+$05] O. Benjelloun, H. Garcia-Molina, J. Jonas, Q. Su, and J. Widom. Swoosh: A Generic Approach to Entity Resolution. Technical report, Stanford University, 2005. (available at `http://dbpubs.stanford.edu/pub/2005-5`).

[BGMK$^+$06] O. Benjelloun, H. Garcia-Molina, H. Kawai, T. E. Larson, D. Menestrina, and S. Thavisomboon. D-Swoosh: A Family of Algorithms for Generic, Distributed Entity Resolution. Technical report, Stanford University, 2006. (available at `http://dbpubs.stanford.edu/pub/2006-8`).

[BSHW06] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. In *Proc. of VLDB (to appear)*, 2006.

[HS95] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proc. of ACM SIGMOD*, pages 127–138, 1995.

[MBGM05] D. Menestrina, O. Benjelloun, and H. Garcia-Molina. Generic Entity Resolution with Data Confidences. Technical report, Stanford University, 2005. (available at `http://dbpubs.stanford.edu/pub/2005-35`).