Evaluating Entity Resolution Results

David Menestrina, Steven Euijong Whang, and Hector Garcia-Molina
Computer Science Department
Stanford University
353 Serra Mall, Stanford, CA 94305, USA
{dmenest,swhang,hector}@cs.stanford.edu

ABSTRACT

Entity Resolution (ER) is the process of identifying groups of records that refer to the same real-world entity. Various measures (e.g., pairwise F_1, cluster F_1) have been used for evaluating ER results. However, ER measures tend to be chosen in an ad-hoc fashion without careful thought as to what defines a good result for the specific application at hand. In this paper, our contributions are twofold. First, we conduct an analysis on existing ER measures, showing that they can often conflict with each other by ranking the results of ER algorithms differently. Second, we explore a new distance measure for ER (called "generalized merge distance" or GMD) inspired by the edit distance of strings, using cluster splits and merges as its basic operations. A significant advantage of GMD is that the cost functions for splits and merges can be configured, enabling us to clearly understand the characteristics of a defined GMD measure. Surprisingly, a state-of-the-art clustering measure called Variation of Information is a special case of our configurable GMD measure, and the widely used pairwise F_1 measure can be directly computed using GMD. We present an efficient linear-time algorithm that correctly computes the GMD measure for a large class of cost functions that satisfy reasonable properties.

1. INTRODUCTION

Entity Resolution (ER) is the problem of identifying groups of records that represent the same real-world entity and then merging the matching records. For example, two companies that merge may want to combine their customer records: for a given customer that dealt with both companies, they create a composite record that combines the known information. In this paper, we will consider the task of evaluating the results of an entity resolution process.

Usually when we compare entity resolution algorithms, we run them on a data set and compare the results to a "gold standard". The gold standard is an entity resolution result that we assume to be correct. In many cases, the gold standard is generated by a group of human experts. On large data sets where the task is too large to be handled by a human, it is not uncommon to run an exhaustive algorithm to generate a result, and treat that result as the gold standard. Then we can compare the results of other approximate or heuristic-based algorithms to this standard in the same manner we would compare them to a human-generated gold standard.

Set	ER Result
Gold Standard	$\{\langle a,b\rangle, \langle c,d\rangle, \langle e,f,g,h,i,j\rangle\}$
R_1	$\{\langle a\rangle, \langle b\rangle, \langle c\rangle, \langle d\rangle, \langle e,f,g,h,i,j\rangle\}$
R_2	$\{\langle a,b\rangle, \langle c,d\rangle, \langle e,f,g\rangle, \langle h,i,j\rangle\}$
R_3	$\{\langle a,b,c,d\rangle, \langle e,f,g,h,i,j\rangle\}$

Table 1: Comparing two ER results

A key component of this type of evaluation is a method of assigning a number to express how close a given ER result is to the gold standard. Many ER measures (e.g., pairwise F_1, cluster F_1) have been proposed for comparing the results of ER algorithms [1, 2, 3], but there is currently no agreed standard measure for evaluating ER results. Most works tend to use one ER measure over another without a clear explanation of why that ER measure is most appropriate. The pitfall of using an arbitrary measure is that different measures may disagree on which ER results are the best.

Let us consider a brief example. Using letters to represent records, consider an entity resolution problem with an input set of records $I = \{a, b, c, d, e, f, g, h, i\}$. Three possible ER results are shown in Table 1, along with the gold standard. We have used angle brackets to denote groups of records that have been determined to refer to the same real-world entity in a result. For example, the algorithm that generated result R_1 decided that records a, b, c, and d all refer to distinct entities, while records e, f, g, h, i, and j all refer to the same entity.

Suppose we are evaluating two ER results R_1 and R_2, against the gold standard G. Using an ER measure that evaluates a result based on the number of record pairs that match, R_1 could be a better solution because it found 15 correct pairs (i.e., all record pairs in $\langle e, f, g, h, i, j\rangle$) while R_2 only found 8 correct pairs. On the other hand, if we use a measure that evaluates results based on correctly resolved entities in the gold standard, R_2 could be considered better than R_1 because R_2 contains two correctly resolved entities $\langle a, b\rangle$ and $\langle c, d\rangle$ while R_1 only has one correct entity $\langle e, f, g, h, i, j\rangle$. As another example, suppose that we compare R_2 and R_3. One measure could be more focused on high precision and prefer R_2 over R_3 because R_2 has only found correctly matching records while R_3 has found some non-matching records (e.g., a and c do not match). On the other hand, another measure might consider recall to be more important and prefer R_3 over R_2 because R_2 has not found all the matching record pairs (unlike R_3).

Surprisingly, such conflicts between ER measures can occur frequently. Section 6.1 thoroughly discusses conflicts and empirically demonstrates the frequency of conflicts. It is tempting to suggest that when conflicts arise, one of measures involved must be faulty in some way. However, since different applications may have different criteria that define the "goodness" of a result, we cannot simply claim one measure to be better than another.

The main contributions of this paper are twofold. We provide a survey of ER measures that have been used to date, experimentally demonstrate the frequency of conflicts between these measures, and provide an analysis of how the measures differ. In studying these measures, we noticed a missing component in the space of existing measures. So the second main contribution of this paper is a new measure for evaluating ER and an exploration of its relationships to other measures.

Our new measure is inspired by the edit distance of strings [4]. Rather than the insertions, deletions and swaps of characters used in edit distance, our measure is based upon the elementary operations of merging and splitting clusters. We therefore call this measure "merge distance". A basic merge distance that simply counts the number of splits and merges may be a good choice for certain applications, but as we have mentioned, no single ER measure is better than all the others. However, if we generalize merge distance by letting the costs of merge and split operations be determined by functions, we arrive at an intuitive, configurable measure that can support the needs of a wide variety of applications. While differently configured merge distance measures may still conflict, we now have a better understanding of what quality each configured measure is evaluating. Surprisingly, at least two state-of-the-art measures are closely related to generalized merge distance: the Variation of Information (VI) [5] clustering measure is a special case of generalized merge distance while the pairwise F_1 [3] measure can be directly computed using generalized merge distance.

We further propose a linear-time algorithm (called Slice) that efficiently computes generalized merge distance for a large class of cost functions that satisfy reasonable properties. As we argue in this paper (Section 5) gold standards can be very large, so computing measures can be expensive, especially with the quadratic algorithms used for many measures. To the best of our knowledge, the Slice algorithm is the first provably scalable algorithm for ER measures. A non-trivial result is that the pairwise F_1 and VI distances can be computed using our Slice algorithm in linear time.

In summary, our contributions are as follows:

- We define our models for ER and ER measures, and conduct an extensive survey on ER measures (Sections 2∼3).
- We propose generalized merge distance (GMD), a new measure that uses the elementary operations of cluster splits and merges to measure the distance from one ER result to another. We propose an efficient linear-time algorithm (called the Slice algorithm) that computes GMD for a large class of cost functions that satisfy reasonable properties (Sections 4∼5).
- We conduct various experiments on ER measures. Although most papers use a single measure to evaluate algorithms, we show that ER measure conflicts can occur frequently in practice where ER algorithms are ranked differently depending on the ER measure. Next, we show how various configurations of GMD can help capture different qualities of ER results. Finally, we demonstrate the scalability of the Slice algorithm (Section 6).

2. FRAMEWORK

An ER algorithm takes as input a set of records I and groups together records that represent the same real world entity. We represent the output of the ER process as a partition of the input. Given an input set of records $I = \{a, b, c, d, e\}$, an output can be $\{\langle a, b, c\rangle, \langle d, e\rangle\}$, where the angle brackets denote the clusters in the partition. For shorthand, we may in some cases represent a clusters as a simple string of records, i.e., the partition $\{\langle a, b, c\rangle, \langle d, e\rangle\}$ can be written as simply $\{abc, de\}$. In this example, two real world entities were identified, with a, b, c representing the first, and d, e representing the second.

To evaluate an ER algorithm, we must compare its result R to a gold standard S, which is also a partition of the input records I. We would like to define a measure $D(R, S)$ that computes the distance between R and S. We assume D to satisfy the following two conditions: $D(R, R) = 0$ for any partition R of I and $D(R, S) \geq 0$ for any two partitions R and S of I.

3. EXISTING MEASURES

There are many measures used in the Information Retrieval (IR) and AI communities that measure the quality of clustering. Evaluating clusters is a broader topic than evaluating ER results because ER is a special case of clustering, in which the clusters tend to be small and items in each cluster are typically quite distinct from items in other clusters [6]. Hence, the ER literature has historically only adopted a small subset of IR clustering measures. We consider the most popular measures used in practice: pF_1, cF_1, K, ccF_1, and VI. The definitions of the measures can be found in Appendix A. In this section, we provide a brief description for each measure:

- pF_1 [3]: Combines the pairwise precision (fraction of record pairs from R that are also in S) and pairwise recall (fraction of S pairs found in R).
- cF_1 [7]: Combines the fraction of clusters from R that are also in S and the fraction of clusters from S in R.
- K [8]: Combines the average similarity of clusters in R with respect to those in S and the average similarity of clusters in S with respect to those in R.
- ccF_1 [9]: Similar to cF_1, but the clusters are now compared by their similarities instead of whether or not they are identical.
- VI [5]: Measures the "information" lost and gained while converting R to S.

4. GENERALIZED MERGE DISTANCE

Although the measures above are widely used, they tend to be used without a clear understanding of what defines a good result for a specific application at hand. In this section, we propose a configurable ER measure that returns the distance between R and S based on two fundamental cluster editing operations: split and merge [10]. Edit distance is a common measure in other domains such as string-to-string matching [4] where the basic operations are inserts, deletes, updates, and swaps. Compared to the other measures, we believe an edit distance approach is a natural way of evaluating an ER result because the splits and merges needed to convert R to S can reflect the amount of work needed to correct R.

A measure based on splits and merges was first proposed by Al-Kamha et al. [11], which we call basic merge distance. The basic merge distance (BMD) measure is defined as the minimum number of cluster splits and merges (where all splits precede any merges) required to modify an ER result R into another result S. In the example from Table 1, only one merge is required to get from R_2 to the gold standard (i.e., $BMD = 1$). Result R_1 is comparatively further away from the gold standard, as $BMD = 2$.

The definition of basic merge distance immediately raises some questions on how we can generalize it. In some cases, we may want to penalize splits more than merges, or vice versa. Further, the "badness" of a split or merge may depend on the sizes of the clusters that are being merged or split. In this section, we define a generalized merge distance (GMD) that creates a larger space of possible measures. We also relax the restriction that all splits must precede any merges. In Section 4.2 we show that the space of configured GMD measures includes distance measures closely related to the pairwise precision and recall measures of Appendix A.1, as well as the VI measure of Appendix A.3.

We first formalize the notions of cluster splits, cluster merges, and legal paths.

Definition 4.1. *A split is an operation $c \to c_1, c_2$ where $c_1 \cap c_2 = \varnothing$, $c_1 \cup c_2 = c$, and $c_1, c_2 \neq \varnothing$. The result of applying a split to a partition P is $(P - \{c\}) \cup \{c_1, c_2\}$. A split is a valid operation on P if and only if $c \in P$.*

Definition 4.2. *A merge is an operation $c_1, c_2 \to c$ where $c = c_1 \cup c_2$. The result of applying a merge to a partition P is $(P - \{c_1, c_2\}) \cup \{c\}$. A merge is a valid operation on P if and only if $c_1, c_2 \in P$.*

As a matter of notation, the result of applying an operation o (which can be either a merge or split) to a partition P can be written $P : o$. Note that the result of an operation on a partition is still a partition, so we may apply operations to a partition in sequence. The application of operations o_1 and o_2 to P in sequence can be written $P : o_1 : o_2$. However, we will use commas to separate operations instead: $P : o_1, o_2$.

We require that the editing of clusters is only done based on the given clustering information in R and S. Specifically, a merge cannot create newly clustered records that are not in the same cluster in S. For example, consider $R = \{\langle a, c\rangle, \langle b, d\rangle\}$ and $S = \{\langle a, b\rangle, \langle c, d\rangle\}$. Notice that by merging $\langle a, c\rangle$ and $\langle b, d\rangle$ into $\langle a, b, c, d\rangle$ and then splitting $\langle a, b, c, d\rangle$ into $\langle a, b\rangle$ and $\langle c, d\rangle$, we only need to do one merge and one split, which is better than splitting $\langle a, c\rangle$ and $\langle b, d\rangle$ into the records a, b, c, d, and then merging a, b into $\langle a, b\rangle$ and c, d into $\langle c, d\rangle$ (i.e., two splits and two merges). However, the first approach creates new clusterings in $\langle a, b, c, d\rangle$ (i.e., a clusters with d, and b clusters with c) that do not appear in the clusters of S, violating our condition. Intuitively, editing R to S requires removing the clustering information found in R only and adding the new information in S. The definition of a legal path captures this idea:

Definition 4.3. *A path from partition R to partition S is a sequence of operations $o_1, o_2, \ldots, o_n$ where $S = R : o_1, o_2, \ldots, o_n$ and o_i is a valid operation on $R : o_1, o_2, \ldots, o_{i-1}$ for all o_i. We say that a path is a* legal *path from R to S if for any operation that is a merge $o_1 = c_1, c_2 \to c$, then there exists a cluster $p \in S$ where $c \subseteq p$.*

Notice that, compared to the BMD measure, we no longer restrict all splits to precede merges, but use a more relaxed condition for legal paths.

We now formalize our GMD measure.

Definition 4.4. *The f_m, f_s generalized merge distance $GMD_{f_m,f_s}(R, S)$ from a partition R to another partition S is the minimum cost of a legal path from R to S, where:*

- *the cost of a merge operation $c_1, c_2 \to c$ is $f_m(|c_1|, |c_2|)$.*
- *the cost of a split operation $c \to c_1, c_2$ is $f_s(|c_1|, |c_2|)$.*

We assume some reasonable properties of the functions f_m and f_s:

1. Operations cannot have negative costs: $f_m(x, y) \geq 0$ and $f_s(x, y) \geq 0$.
2. The cost functions are symmetric: $f_m(x, y) = f_m(y, x)$ and $f_s(x, y) = f_s(y, x)$.
3. The cost functions monotonically increase with their parameters: $f_m(x, y) \leq f_m(x + j, y + k)$ and $f_s(x, y) \leq f_s(x + j, y + k)$ for non-negative j, k.

Given the above three properties, we can prove there exists a minimum cost legal path from a partition R to a partition S where all of the split operations precede the merge operations. This result vastly reduces the search space for a minimum cost path and thus leads to an efficient algorithm for computing GMD.

Theorem 4.1. *For any partitions R and S, there exists a minimum cost legal path from R to S where all split operations precede all merge operations.*

Proof. The proof is available in the extended version of this paper [12]. Here we only informally sketch the key idea. We start with a minimum cost legal path p, and transform it a step at a time by moving split operations towards the start of p and merge operations towards the end, until we get the desired path p'. □

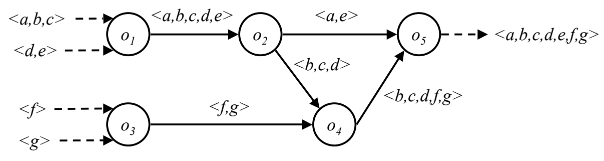


Figure 1: A precedence graph.

To perform the transformations, we view the path as a graph, as illustrated in Figure 1. For instance, operation o_2 is a split that yields clusters $\langle a, e\rangle$ and $\langle b, c, d\rangle$. When we swap merge o_1 with split o_2 (to have splits ahead of merges), we obtain the graph of Figure 2.

Note that the merge (o_1) followed by the split (o_2) has different results from the split (o_1') followed by the merge

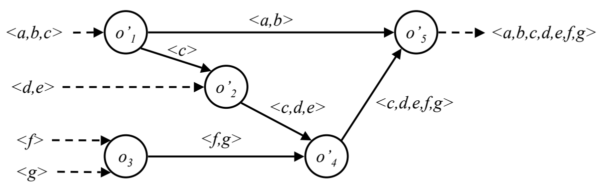


Figure 2: Precedence graph for the transformed path.

(o'_2) (the former yields $\langle a,e\rangle, \langle b,c,d\rangle$ while the latter yields $\langle a,b\rangle, \langle c,d,e\rangle$). However, the resulting cluster sizes are the same (i.e., 2 and 3, respectively), which keeps the cost of subsequent operations the same. Because the results are different, the operations downstream need to be adjusted, again keeping the sizes of all clusters unchanged. The fact that all clusters are subsets of the final cluster $\langle a,b,c,d,e,f,g\rangle$ (path p is legal) ensures that the resulting path p' gives the same result as p. [1]

We will use the term "splits-first path" to refer to a path with all split operations preceding all merge operations. We note that any splits-first path from R to S is also a legal path, because if there was an operation that merged two clusters that are not merged in S, then subsequent operations cannot be splits, and thus the result of the path could not be S.

4.1 Operation Order Independence

If functions f_s and f_m have a property we call *operation order independence*, then it turns out there is an efficient way to compute the GMD between two partitions. This property makes GMD intuitive in a sense that a series of splits (merges) on a given set of records costs the same regardless of the ordering of operations. We first define order independence and then discuss how it helps.

DEFINITION 4.5. *We say that a function f is* operation order independent *if it satisfies $f(x,y)+f(x+y,z) = f(x,z)+f(x+z,y)$ for all x,y,z.*

We call this property operation order independence because it implies that the order in which certain operations are performed is unimportant. Suppose that we wish to merge three clusters c_x, c_y, and c_z (with sizes x, y, and z, respectively) all together into a single cluster. If we merge c_x and c_y together first, and then merge the resulting cluster with c_z, observe that the resulting cost would be given by the left-hand side of the equation in Definition 4.5. The cost of merging c_x and c_z together first, and then merging c_y with the result is given by the right-hand side of the equation, and therefore with operation order independence, these two paths would have the same cost.

The class of operation order independent functions has been studied in [13], which shows there is a vast number of such functions. We believe that three classes of such functions are sufficient for our needs: $f(x,y) = k$, $f(x,y) = kxy$, and $f(x,y) = k_1 + k_2xy$. (One can easily verify the property holds for these classes by plugging them into the equation in Definition 4.5.) The first class can be used to model merges or splits with cost independent of the cluster size, and can be used to emulate the BMD measure. The second class can be used when we want to penalize operations on large clusters, e.g., when failing to merge two large clusters is worse that failing to merge two small clusters. The second class of measures can be used to directly compute (see Section 4.2) the pairwise precision and recall measures of Appendix A.1. The third class of function is a blend of the first two.

[1]If o_5 were removed from p, then the swap would give us a path with a different result from p. This is because without o_5, path p would not be a legal path, since o_1 would be an illegal merge.

When f_s and f_m are operation order independent, we can easily construct a splits-first legal path that has minimum cost. We call this path a "bare necessities" path because it first breaks down the initial set R into the bare "fragments" that are needed for building up S. For example, say that $R = \{\langle a,b,c\rangle, \langle d,e,f\rangle\}$ and $S = \{\langle a,b,d\rangle, \langle c,e,f\rangle\}$. In this case, the bare necessities path from R to S would first split the R clusters into $\langle a,b\rangle, \langle c\rangle, \langle d\rangle, \langle e,f\rangle$, i.e., into the fragments that are needed for then putting together the clusters in S. Actually, there are multiple bare necessities paths, corresponding to the different ways one can split R into the bare fragments and then merge them into S.

THEOREM 4.2. *If both f_m and f_s are operation order independent, then any bare necessities path from R to S is a minimum cost legal path from R to S.*

PROOF. Due to space constraints, the proof is available only in the extended version of this paper [12]. ☐

Our Slice algorithm (see Section 5) computes GMD by computing the cost of a bare necessities path. Because such a path has a simple structure, the computation of its cost turns out to be efficient.

4.2 Relationship to Other Measures

Several other measures are closely related to GMD. First, the BMD measure in [11] is identical to the GMD measure when $f_m(x,y) = f_s(x,y) = 1$. Second, the VI measure (Appendix A.3) is a special case of GMD where f_m and f_s are chosen as follows:

THEOREM 4.3. *$VI(R,S) = GMD(R,S)$ when $f_m(x,y) = f_s(x,y) = h(x+y) - h(x) - h(y)$, with $h(x) = \frac{x}{N}\log\frac{x}{N}$ where N is the total number of records in I.*

PROOF. Due to space constraints, the proof is available only in the extended version of this paper [12]. ☐

Third, the pF_1 distance can be computed directly using GMD. The theorem below shows how to compute pairwise precision and pairwise recall using GMD. The pF_1 distance is then the harmonic mean of the two values. We use the symbol $\perp$ to refer to a partition with each record alone in its own cluster.

THEOREM 4.4. *$PairPrecision(R,S) = 1 - \frac{GMD(R,S)}{GMD(R,\perp)}$ when $f_m(x,y) = 0$ and $f_s(x,y) = xy$. $PairRecall(R,S) = 1 - \frac{GMD(R,S)}{GMD(\perp,S)}$ when $f_m(x,y) = xy$ and $f_s(x,y) = 0$.*

PROOF. The full proof has been removed due to space constraints and is available in the extended version of this paper [12]. The intuition behind the proof is that when a cost function is set to xy, the cost of a split or merge is equal to the number of pairs that are removed or added. ☐

The various relationships are possible because of the configurability of GMD. Since the f_m and f_s functions used in this section are all operation order independent, we can use the linear time Slice algorithm described in the next section to compute all the measures above. This is exciting, especially because the straightforward implementation of pF_1 and VI would be quadratic in the worst case.

4.3 Configuring the Cost Functions

We have argued that a configurable cost metric is desirable because it lets an application specialist describe the types of ER results that are desirable. For instance, is it worse to have a result where a group of records representing the same entity is split into several clusters, or is it worse to have a cluster that mixes several real entities? We have also argued that three simple classes of functions ($f(x, y) = k, kxy$ or $k_1 + k_2xy$) are sufficient to describe the desirability of most ER results.

But do even these limited choices provide too much flexibility, so it becomes difficult to properly configure a metric? First, we believe that it is good to "force" the specialist to think about what is desired, since the configurable metric can guide the ER process, leading to better results. Second, there are relatively simple ways to configure the metric. One strategy, for instance, is to have the specialist provide a set of sample resolutions for a set of records. These examples can show a variety of ways resolution went wrong in the past (e.g., missing a critical merge, or adding an incorrect record to a cluster). Then the specialist can rank these examples, indicating which outcomes are worse than others. Then based on the examples, we can use machine learning to select the class of functions and its parameters that yield the same ordering.

The GMD measure can also be configured in an ad hoc fashion, penalizing merges or splits based on what errors are less desirable, and using the function $f(x, y) = xy$ if the size of the erroneous clusters is important. In Appendix B we discuss several scenarios, and illustrate how a specialist might make choices. Although the process is not fool-proof, it can easily be tuned as ER results are produced.

5. COMPUTING MEASURES

Computing measures efficiently is important because the number of entities to resolve can be huge. Although human-generated gold standards will rarely exceed thousands of records, other gold standards are automatically generated and could result in larger numbers of records. For example, blocking techniques [14] are commonly used to make ER scalable by dividing the data into (possibly overlapping) blocks and only comparing records within the same block, assuming that records in different blocks are unlikely to match. Since blocking techniques may miss matching records, their results are compared with an "exhaustive" ER solution without blocking, which is considered as the gold standard [15]. While large exhaustive ER results may be very expensive to generate, it need only be generated once, whereas the computation of the distance measure will be performed multiple times for a diverse set of blocking algorithms and parameters. The distance computation can therefore take a great deal of time, and a more efficient algorithm provides practitioners more time to tune their algorithms (e.g., experiment with different matching thresholds) over a wide range of options.

Many measures take (or appear to take) quadratic time for computation, which could be prohibitive. For example, a straightforward implementation of the pF_1 measure requires a quadratic number of record pairs to be compared against the actual matching pairs. Similarly, the K measure sums the similarities of all pairs of clusters need to be computed, requiring quadratic time computation. The ccF_1 measure finds the the closest clusters for all clusters and requires a quadratic number of cluster comparisons because finding each closest cluster requires a linear scan of the other ER result in the worst case.

To the best of our knowledge, the topic of efficiency of measure computation is not discussed in any ER paper (there are many works that focus on the efficiency of ER algorithms). Fortunately in this paper, we propose an efficient algorithm (called Slice) that computes GMD in linear time when f_m and f_s are operation order independent functions. The full code can be found in Appendix C. We also show that the pF_1 and VI measures can be computed in linear time using our algorithm. It is an open question if there are linear algorithms for the ccF_1 and K measures.

6. EXPERIMENTS

As mentioned in Section 1, there are two aspects to our paper: a general analysis of ER measures, and the proposal of a generalized merge distance measure. Accordingly, our evaluation mirrors these two aspects. The first part of the experiments show that measure conflicts can easily occur among different ER measures. Hence, simply choosing any ER measure for comparing the accuracy of ER algorithms could be problematic. The second part demonstrates how various configurations of the GMD measure can help capture different qualities of ER results. We then demonstrate the runtime performance of the Slice algorithm.

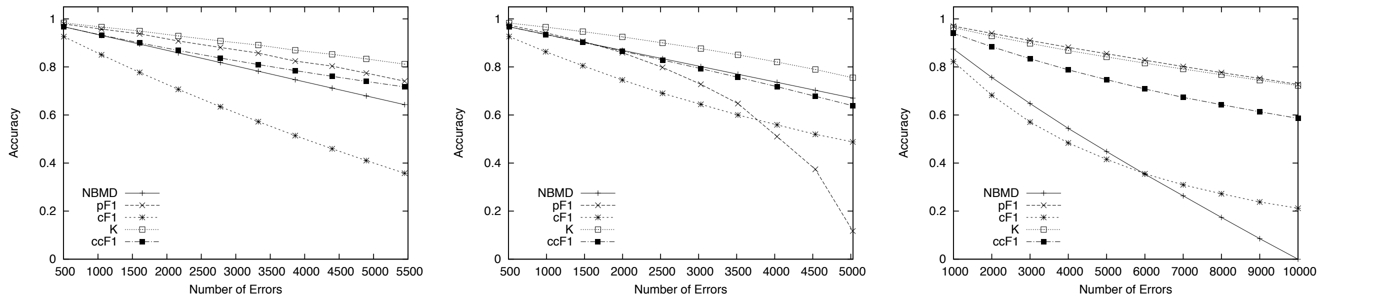
To study the ER measures, we need an ER result R and gold standard G. We use both synthetic and real data. To get real data, we ran two ER algorithms – Swoosh [9] and Monge Elkan (ME) [16] – on a comparison shopping dataset provided by Yahoo! Shopping and a hotel dataset provided by Yahoo! Hotel. Further details on our data and ER algorithms can be found in Appendix D.1.

6.1 Measure Conflicts

Most of the papers surveyed in Section 3 use a single measure (or two closely related measures, e.g., pairwise precision and pairwise recall) to evaluate algorithms. In this section we show that such a unilateral evaluation can be problematic, since different measures can lead to different rankings of algorithms. That is, measures can "conflict." For each measure M, we define a function $isBetter(M, R_1, R_2)$ that is *true* if R_1 is significantly better than R_2 according to M. For GMD, this function can be defined as $GMD(R_1) < GMD(R_2) - \epsilon$. For any other measure M that returns an accuracy value instead of a distance, *isBetter* can be $M(R_1) > M(R_2) + \epsilon$. The constant ϵ is chosen such that *isBetter* returns *true* only if the measure difference is non trivial. In our experiments, we set $\epsilon = 0.01$ for all measures. We now define a measure conflict:

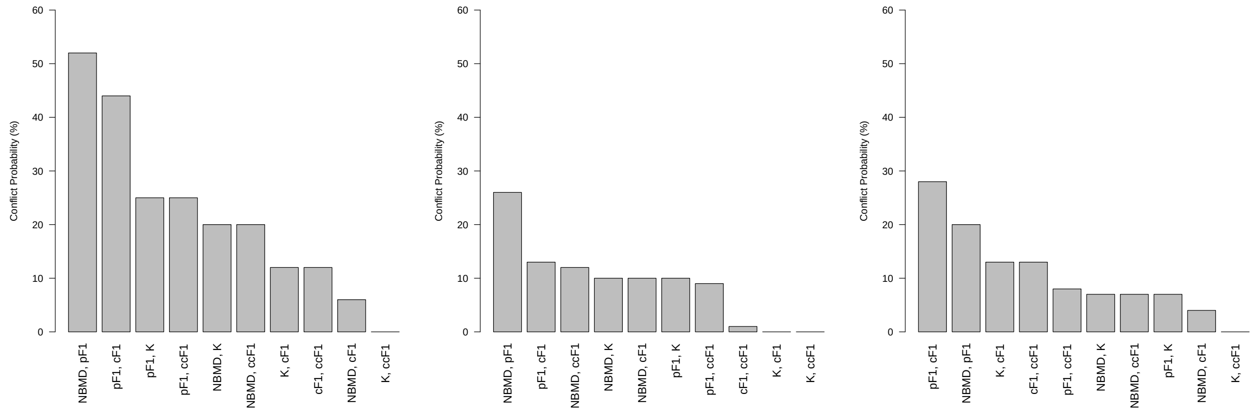
Definition 6.1. *Two measures M_1 and M_2* conflict *when, given two algorithms A_1 and A_2 that produce the ER results R_1 and R_2, respectively, $isBetter(M_1, R_1, R_2) = true$ and $isBetter(M_2, R_2, R_1) = true$.*

Measure conflicts occur because different measures evaluate different aspects of ER results. For example, pairwise precision only measures the portion of correctly matching record pairs among the result while pairwise recall measures the portion of all correctly matching pairs found in the result. Similarly, the ER measures we implement have different sensitivities to the various aspects of ER results.



(a) Broken entity errors (b) Glued entity errors (c) Misplaced entity errors

Figure 3: Sentivity comparison for single error types



(a) Glued and misplaced entity errors (b) Broken and misplaced entity errors (c) Broken and glued entity errors

Figure 4: Conflict frequencies

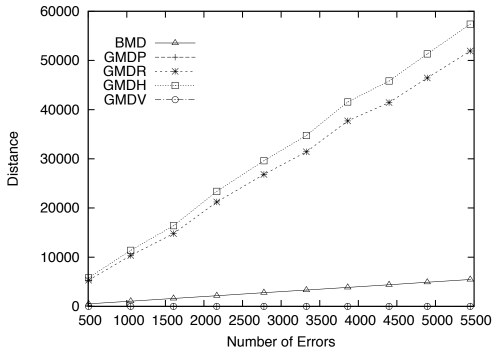
To see how frequently conflicts could occur, we first measure how "sensitive" the ER measures are for each error type in Figure 3 (the definitions for each error type can be found in Appendix D.1). We used $E = 10{,}000$ entities and the default Zipfian exponent $e = 1.5$ to generate the gold standard G. For each error type, we generated 10 ER results with increasing numbers of errors. We evaluated each ER result with pF_1, cF_1, K, ccF_1, and a normalized version of BMD (which we refer to as $NBMD$) that returns an accuracy value within the range [0, 1]. The $NBMD$ between an ER result R and the gold standard G is defined as $1 - \frac{BMD(R,G)}{BMD_{max}}$ where BMD_{max} is the largest BMD among the ER results against S for all the three experiments in Figure 3. As a result, the $NBMD$ value in one of the plots (in this case Figure 3(c)) is 0 for the largest number of errors. For each error type, all the accuracy values of the measures are monotonically decreasing as the number of errors increases. The number of errors (x-axis) is defined as $|R|$-$|G|$ for ER results with broken entity errors, $|G|$-$|R|$ for ER results with glued entity errors, and the number of records misplaced for ER results with misplaced entity errors.

One can see from Figure 3 that if we confine ourselves to a single error type, any one measure is good enough to evaluate ER results. That is, if R_2 has more errors than R_1, then $isBetter(M, R_1, R_2) = true$ for all measures M. In other words, there are no conflicts with unimodal errors.

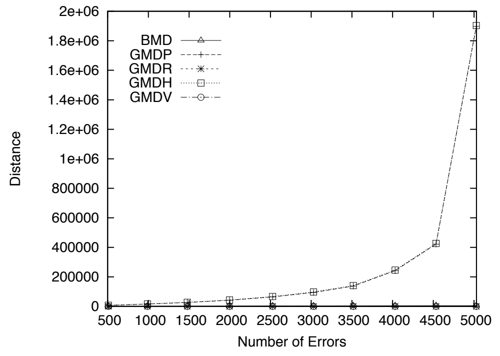
However, by comparing the accuracy values for ER results with different types of errors, we can identify many conflicts. For example, say Algorithm 1 produces many broken entities, and its result R_1 contains 5,450 errors (right most data points in Figure 3(a)). While resolving the same input set, Algorithm 2 generates many glued entities, and its result R_2 contains 5,027 errors (right most data points in Figure 3(b)). According to ccF_1, $isBetter(ccF_1, R_1, R_2) = true$ because $ccF_1(R_1) = 0.72$ while $ccF_1(R_2) = 0.64$. On the other hand, for cF_1, $isBetter(cF_1, R_2, R_1) = true$ because $cF_1(R_2) = 0.49$ while $cF_1(R_1) = 0.36$. As a result, ccF_1 and cF_1 conflict on R_1 and R_2, i.e., ccF_1 tells us Algorithm 1 is better, while cF_1 tells us Algorithm 2 is better!

Figure 4 shows the number of conflicts that occur for each measure pair based on the data in Figure 3, sorted in decreasing numbers of conflicts. Each plot compares ER results of one error type to the ER results of another error type. Since there are 10 ER results for each error type according to Figure 3, we compare all 10×10 ER result pairs for each pair of ER measures. For example, between the $NBMD$ and pF_1 measures, we found 52 conflicts among the 100 ER result pairs (hence the 52% conflict probability in the first plot of Figure 4). Overall, the $NBMD$ and pF_1 measures conflict more frequently than other pairs of measures. The average conflict probability of two measures was 21.6%, 9.1%, and 10.7% for the three plots, respectively. Using these results, we can compute the average conflict probability for all ER result pairs (i.e., including the pairs that have the same error type) as $\frac{(21.6+9.1+10.7)\times 100}{3\times 100+3\times\binom{10}{2}} = 9.5\%$. Hence, the chance of measure conflicts is clearly not trivial.

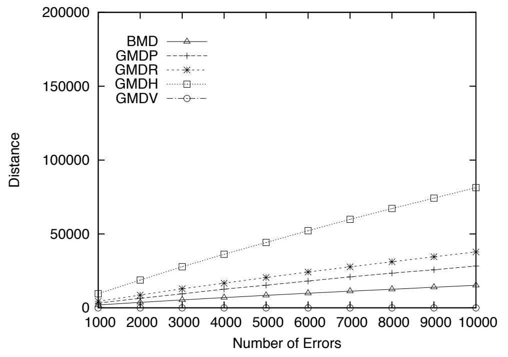
We show in Appendix D.2 that that conflicts can also occur in real world applications, and evaluations of ER algorithms need to consider multiple measures (something that to date is seldom done).



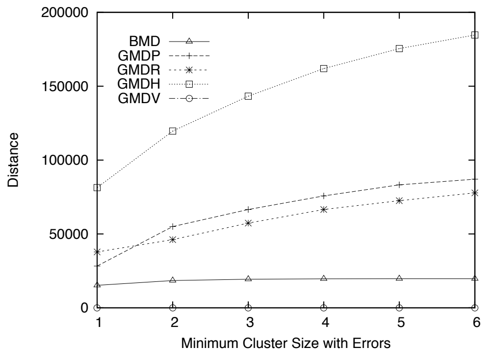
(a) Broken entity errors



(b) Glued entity errors



(c) Misplaced entity errors



(d) Misplaced entity errors in large clusters

Figure 5: Sensitivities of GMD measures for different error types

6.2 Configured GMD Results

We experiment on various configurations of the GMD measure to capture different qualities of ER results. In Section 4.3, we discussed how the cost functions of a GMD measure can be configured based on how much their are sensitive to cluster sizes and whether one operation (merge or split) should be more expensive than the other operation. Table 2 shows five different configurations of the GMD measure we will use in our experiments. The BMD measure assumes constant costs for merges and splits. The GMD_P measure can be used to compute pairwise precision (see Theorem 4.4). Similarly, the GMD_R measure can be used to compute pairwise recall. A harmonic mean of pairwise precision and pairwise recall computes pF_1. The GMD_H measure has hybrid cost functions that have high sensitivities to record sizes as well as constant overheads for each merge and split. Finally, the GMD_V measure is equivalent to the VI measure.

Table 2: Configured GMD measures

	BMD	GMD_P	GMD_R	GMD_H	GMD_V
f_m	1	0	xy	$xy+1$	$h(x{+}y)$-$h(x)$-$h(y)$[a]
f_s	1	xy	0	$xy+1$	$h(x{+}y)$-$h(x)$-$h(y)$

[a] $h(z) = \frac{z}{N}\log\frac{z}{N}$ where N is the total number of records.

Figure 5 compares the sensitivities of the configured GMD measures in Table 2. We used the same gold standard and ER results used for Figure 3. Figures 5(a) and 5(b) demonstrate sensitivities to broken entity errors and glued entity errors, respectively. The plots of GMD_P (using the symbol $+$) and GMD_R (using the symbol $*$) switch places because of their different sensitivities to the two types of errors, while the relative ordering of the other plots remain the same. Figure 5(c) shows that for misplaced entity errors, the sensitivities of the five configured GMD measures do not differ much because there is an even mix of broken and glued entity errors. However, Figure 5(d) shows how the sensitivities to record size vary among the configured GMD measures. While adding the same 10,000 misplaced entity errors to each ER result, we increased the minimum size of clusters that could contain misplaced entity errors from 1 to 5. The higher the minimum size, the more "concentrated" the errors are in large clusters. (Notice that when the minimum size is 1, the GMD results are identical to the right-most points in Figure 5(c).) As a result, the GMD_P, GMD_R, GMD_H measures, which have the most expensive cost functions, show substantial increases in distances when the errors are concentrated in large clusters. The GMD_V measure, which has logarithmic cost functions, is moderately sensitive (although not clearly shown in the plot due to its small distances) while the BMD measure is the least sensitive.

We now show how real world algorithms can be compared using the configured GMD measures. Using the measures in Table 2, we were able to do a detailed comparison between the Swoosh and ME algorithms. While the Swoosh algorithm is superior to the ME algorithm in terms of broken entity errors, the ME algorithm has fewer broken entity errors. Also, the Swoosh algorithm does not resolve large clusters well (more details can be found in Appendix D.3).

6.3 Runtime Performance

As discussed in Section 5, ER datasets can be huge, and the computation times for measures can be very significant. We compared the computation times for the BMD, pF_1,cF_1,K,ccF_1, and VI measures. Our results show that any implementation using the Slice algorithm is scalable to large ER results, with runtime increasing linearly by the number of entities (more details can be found in Appendix D.4).

7. CONCLUSION

We have proposed an edit distance measure for ER (called "generalized merge distance" or GMD) that computes the shortest edit distance from an ER result to a gold standard using merges and splits as the basic operations on clusters. A powerful feature is that the merge and split costs can be configured based on record sizes. We proposed an efficient algorithm (called Slice), which computes GMD in linear time for a large class of merge and split cost functions. Interestingly, the state-of-the-art VI clustering measure is a special case of GMD, and the dominantly used pF_1 measure for ER can be directly computed using GMD. As a result, both VI and $pF1$ can be computed efficiently using our Slice algorithm.

We have shown in our experiments that evaluating ER algorithms based on a single ER measure is problematic because different measures conflict with each other. Such conflicts occur because each measure focuses on certain features in the ER results for computing accuracy. We demonstrated that, by using the configured GMD measures, one could more precisely evaluate a given application. Finally, we have demonstrated that the Slice algorithm is scalable and can be used to evaluate very large datasets. Thus, we believe that the GMD measure fills a hole in the space of available ER measures, and that it clarifies the relationship between the available ER measures.

There are interesting open issues for the GMD measure. We have already shown that the pF_1 and VI measures are closely related to GMD. We believe that edit distance measures for ER and clustering have yet to be fully explored and suspect that GMD could be a fundamental way of generating ER and in general clustering measures.

8. REFERENCES

[1] W. Winkler, "Overview of record linkage and current research directions," Statistical Research Division, U.S. Bureau of the Census, Washington, DC, Tech. Rep., 2006.

[2] A. H. F. Laender, M. A. Gonçalves, R. G. Cota, A. A. Ferreira, R. L. T. Santos, and A. J. C. Silva, "Keeping a digital library clean: new solutions to old problems," in *ACM Symposium on Document Engineering*, 2008, pp. 257–262.

[3] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval.* Cambridge University Press, July 2008.

[4] R. A. Wagner, "On the complexity of the extended string-to-string correction problem," in *STOC*, 1975, pp. 218–223.

[5] M. Meila, "Comparing clusterings by the variation of information," in *COLT*, 2003, pp. 173–187.

[6] A. K. McCallum, K. Nigam, and L. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *Proc. of KDD*, Boston, MA, 2000, pp. 169–178.

[7] J. Huang, S. Ertekin, and C. L. Giles, "Efficient name disambiguation for large-scale databases," in *PKDD*, 2006, pp. 536–544.

[8] R. G. Cota, M. A. Gonçalves, and A. H. F. Laender, "A heuristic-based hierarchical clustering method for author name disambiguation in digital libraries," in *SBBD*, 2007, pp. 20–34.

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

[10] M. Meila, "Comparing clusterings: an axiomatic view," in *ICML*, 2005, pp. 577–584.

[11] R. Al-Kamha and D. W. Embley, "Grouping search-engine returned citations for person-name queries," in *WIDM*, 2004, pp. 96–103.

[12] D. Menestrina, S. E. Whang, and H. Garcia-Molina, "Evaluating Entity Resolution Results (Extended version)," Stanford University, Tech. Rep., available at http://ilpubs.stanford.edu/930/.

[13] M. Hosszú, "On the functional equation f(x+y,z) + f(x,y) = f(x,y+z) + f(y,z)," *Periodica Mathematica Hungarica*, vol. 1, no. 3, pp. 213–216, 09 1971.

[14] H. B. Newcombe and J. M. Kennedy, "Record linkage: making maximum use of the discriminating power of identifying information," *Commun. ACM*, vol. 5, no. 11, pp. 563–566, 1962.

[15] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina, "Entity resolution with iterative blocking," in *SIGMOD Conference*, 2009, pp. 219–232.

[16] A. E. Monge and C. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records," in *DMKD*, 1997, pp. 23–29.

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

[18] L. Jin, C. Li, and S. Mehrotra, "Efficient record linkage in large data sets," in *DASFAA*, 2003, pp. 137–.

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

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

[21] M. Elfeky, V. Verykios, and A. Elmagarmid, "TAILOR: A record linkage toolbox," in *ICDE*, 2002.

[22] R. Baxter, P. Christen, and T. Churches, "A comparison of fast blocking methods for record linkage," in *Proc. of ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[23] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee, "Framework for evaluating clustering algorithms in duplicate detection," *PVLDB*, vol. 2, no. 1, pp. 1282–1293, 2009.

[24] M. Bilenko and R. Mooney, "Adaptive duplicate detection using learnable string similarity measures," in *KDD*, 2003.

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

APPENDIX

A. EXISTING MEASURES

In the body of the paper, we refer to various measures used in the literature: pF_1, cF_1, K, ccF_1, and VI. In this section, we briefly describe these measures.

A.1 Pairwise Comparison

The pairwise comparison approach counts the number of pairs of records to evaluate ER results. To define pairwise measures, we define a function $Pairs(P)$ that takes in a partition P and returns the set of distinct pairs of records that are in the same cluster in P. For example, if $P = \{\langle a,b,c\rangle, \langle d,e\rangle\}$, then $Pairs(P) = \{(a,b),(b,c),(a,c),(d,e)\}$. We can now define the similarity measures pairwise precision and pairwise recall:

$$PairPrecision(R,S) = \frac{|Pairs(R) \cap Pairs(S)|}{|Pairs(R)|}$$

$$PairRecall(R,S) = \frac{|Pairs(R) \cap Pairs(S)|}{|Pairs(S)|}$$

A number of ER papers [17, 18] use pairwise precision and pairwise recall to evaluate ER results while earlier works [19, 20] use the rate of false positives (i.e., 1-$PairPrecision(R,S)$) and the rate of false negatives (i.e., 1-$PairRecall(R,S)$) for evaluation. A few works [21, 22] use a variant of pairwise recall while taking into account the reduced number of record comparisons due to blocking techniques. Another work [23] uses a variant of pairwise precision where precision is penalized based on the difference of $|R|$ and $|S|$.

pF_1. The pairwise F_1 measure [3, 24, 25, 6, 2] is the dominant measure in the ER literature and is defined as the harmonic mean of pairwise precision and pairwise recall:

$$pF_1(R,S) = \frac{2 \times PairPrecision(R,S) \times PairRecall(R,S)}{PairPrecision(R,S) + PairRecall(R,S)}$$

For example, if $R=\{\langle a,b\rangle, c, d\}$ and $S=\{\langle a,b\rangle, \langle c,d\rangle\}$, $Pr = \frac{1}{1}$ and $Re = \frac{1}{2}$, making the pairwise $F_1 = \frac{2\times 1\times(1/2)}{1+(1/2)} = \frac{2}{3} =$ 66.67%.

A.2 Cluster-level Comparison

The cluster-level comparison approach sums the similarity of clusters to evaluate ER results instead of counting pairs of records.

cF_1. The cluster F_1 measure [7, 2] counts clusters that exactly match and is defined as the harmonic mean of the cluster precision and cluster recall. The cluster precision is defined as $\frac{|R\cap S|}{|R|}$ while the cluster recall is defined as $\frac{|R\cap S|}{|S|}$. Notice that we are now comparing R and S at the cluster level instead of the record level as in pF_1. Returning to our previous example where $R=\{\langle a,b\rangle, c, d\}$ and $S=\{\langle a,b\rangle, \langle c,d\rangle\}$, the precision is $\frac{1}{3}$ while the recall is $\frac{1}{2}$ because exactly one cluster matches among three clusters in R and two clusters in S. The Cluster F_1 is thus $\frac{2\times(1/3)\times(1/2)}{(1/3)+(1/2)} = \frac{2}{5} = 40\%$. We denote the cluster F_1 measure as cF_1.

K. The K measure [8, 2] sums the similarities of all cluster pairs and is defined as the geometric mean of the Average Cluster Purity (ACP) and the Average Author Purity (AAP). (Here, Author can be thought of as a cluster in the gold standard.) The ACP is defined as $\frac{1}{N}\Sigma_{r\in R}\Sigma_{s\in S}\frac{|r\cap s|^2}{|r|}$ where N is the number of records. (Notice that the records r and s are considered as sets of records.) Similarly, the AAP is defined as $\frac{1}{N}\Sigma_{s\in S}\Sigma_{r\in R}\frac{|r\cap s|^2}{|s|}$. The K measure is then $\sqrt{ACP \times AAP}$. For example, the ACP value for R and S is $\frac{(2^2/2)+(1^2/2)+(1^2/2)}{4} = \frac{3}{4}$ while the AAP value is $\frac{(2^2/2)+(1^2/1)+(1^2/1)}{4} = 1$, making the K value $\sqrt{\frac{3}{4}\times 1}$= 86.6%.

ccF_1. The closest cluster F_1 measure [9] sums the similarities of all "closest" cluster pairs and is defined as the harmonic mean of the closest cluster precision and closest cluster recall values. The closest cluster precision is defined as $\frac{\Sigma_{r\in R}\max_{s\in S}(J(r,s))}{|R|}$ where $J(r,s)$ is the Jaccard similarity $\frac{|r\cap s|}{|r\cup s|}$. The closest cluster precision is thus the sum of the maximum Jaccard similarity coefficients for all r's divided by $|R|$. Similarly, the closest cluster recall is defined as $\frac{\Sigma_{s\in S}\max_{r\in R}(J(s,r))}{|S|}$. For example, the closest cluster precision for R against S is $\frac{(2/2)+(1/2)+(1/2)}{3} = \frac{2}{3}$ while the closest cluster recall is $\frac{(2/2)+(1/2)}{2} = \frac{3}{4}$, making the closest cluster $F_1 = \frac{2\times(2/3)\times(3/4)}{(2/3)+(3/4)} = \frac{12}{17} = 70.59\%$. We denote closest cluster F_1 as ccF_1. Reference [23] uses a variant of ccF_1 that uses a different similarity equation and gives weights to the coefficients when adding them.

A.3 Edit Distance

The closest related work to an edit distance measure for clustering is a state-of-the-art measure called Variation of Information [5] (VI) where we measure the "information" lost and gained while converting one clustering to another as follows:

$$VI(R,S) = H(R) + H(S) - 2I(R,S)$$

Functions H and I represent, respectively, the total entropy of the individual clusters and the mutual information between R and S. (N is the total number of records in I.)

$$H(R) = -\sum_{r\in R}\frac{|r|}{N}\log\frac{|r|}{N}$$

$$I(R,S) = \sum_{r\in R}\sum_{s\in S}\frac{|r\cap s|}{N}\log\frac{|r\cap s|\times N}{|r|\times|s|}$$

B. CONFIGURING THE COST FUNCTIONS

In the body of the paper, we presented various configurations for the GMD measure. In this section, we provide four scenarios where certain configurations of GMD are suitable for properly evaluating ER algorithms. Of course, most applications would favor a perfect ER result. However, given that ER results are not always accurate, applications may be willing to focus on certain qualities while sacrificing others.

Large clusters matter. Suppose that we are interested in the most famous individuals on various social websites (e.g., Facebook, MySpace, etc.). In this case, we might be interested in focusing on the people that have many profiles and are highly active on the Web. When resolving people records on the various social websites, we might thus be more interested in correctly resolving individuals with lots of information (i.e., many records) rather than inactive individuals with only a few records. A reasonable setting could be $f_m = xy$ and $f_s = xy$ because we want to make sure the large clusters are resolved correctly.

All clusters matter. Suppose that we have a shopping application that needs to resolve a large set of product items and display them to users online. Since each item is equally important, we might want a uniform evaluation on all clusters regardless of their sizes. For example, a book that has one record for its description is equally important as an iPod that has ten varying descriptions. Hence, it makes sense to give equal cost to a merge or split for any cluster sizes involved. A reasonable setting then could be $f_m = 1$ and $f_s = 1$ because every cluster is equally evaluated.

Minimize human effort. If a human expert needs to correct the ER results, then merging two clusters could be easier than splitting a single cluster. While merging two clusters does not require much work for a person, splitting requires one to decide which records will be separated with others. The splitting becomes more difficult as the cluster size increases. As a result, we might want to penalize large clusters that are incorrect and need to be split. Thus by setting $f_m = 1$ and $f_s = xy$, we can favor ER results that minimize the splitting effort of the human expert (of course, now the human may have to do more merges).

Complete information for large clusters. Suppose that we are analyzing how popular news spreads through the Web. In this case, we might want to make sure we have all the related webpages for each popular topic even if we also end up gathering irrelevant webpages as well. At the same time, we might not be interested in topics that are not widely disseminated and thus only appear in few webpages. By setting $f_m = xy$ and $f_s = 1$, we can favor ER results that have high recall on large clusters (i.e., popular topics).

Algorithm 1 Slice algorithm

```
Input: $R, S$: the result and gold standard.
$R_i$ and $S_i$ refer to the $i$th clusters of $R$ and $S$ respectively
$f_m, f_s$: the cost functions for merge and split operations
Output: the $f_m, f_s$ merge distance from $R$ to $S$
1: MergeDistance($R, S$)
2: // build a map $M$ from record to cluster number
3: // and store sizes of each cluster in $R$
4: for all $R_i \in R$ do
5:     for all $r \in R_i$ do
6:         $M[r] \leftarrow i$
7:     end for
8:     $Rsizes[i] \leftarrow |R_i|$
9: end for
10: // begin computing cost
11: $cost \leftarrow 0$
12: for all $S_i \in S$ do
13:     // determine which clusters in $R$ contain the records in $S_i$
14:     $pMap \leftarrow \{\}$
15:     for all $r \in S_i$ do
16:         // if we haven't seen this $R$ cluster before, add it to the map
17:         if $M[r] \notin keys(pMap)$ then
18:             $pMap[M[r]] \leftarrow 0$
19:         end if
20:         // increment the count for this partition
21:         $pMap[M[r]] \leftarrow pMap[M[r]] + 1$
22:     end for
23:     // compute cost to generate $S_i$
24:     $SiCost \leftarrow 0$
25:     $totalRecs \leftarrow 0$
26:     for all $(i, count) \in pMap$ do
27:         // add the cost to split $R_i$
28:         if $Rsizes[i] > count$ then
29:             $SiCost \leftarrow SiCost + f_s(count, Rsizes[i] - count)$
30:         end if
31:         $Rsizes[i] \leftarrow Rsizes[i] - count$
32:         if $totalRecs \neq 0$ then
33:             // cost to merge into $S_i$
34:             $SiCost \leftarrow SiCost + f_m(count, totalRecs)$
35:         end if
36:         $totalRecs \leftarrow totalRecs + count$
37:     end for
38:     $cost \leftarrow cost + SiCost$
39: end for
40: return $cost$
```

C. THE SLICE ALGORITHM

In the body of the paper, we motivated the use of an efficient algorithm for computing GMD. In this section, we propose the Slice algorithm, which computes GMD when f_m and f_s are operation order independent functions. The algorithm computes the cost of an arbitrarily selected bare necessities path, and therefore produces the correct answer only when the split and merge cost functions are order operation independent.

The algorithm takes two partitions R and S, as well as the functions f_m and f_s as input. The output of the algorithm will be the f_m, f_s merge distance from R to S (as long as f_m and f_s are operation order independent). The gist of this algorithm is to find the cost to build each cluster $S_i \in S$ by breaking off pieces from clusters in R and then merging them together. We can find the cost to build each S_i independently and then compute the sum for the total cost to move from R to S.

We now explain the details of the algorithm, and execute it over an example input. For the purposes of the example, let $R = \{\langle a, c, e\rangle, \langle b, d, f\rangle\}$ and $S = \{\langle a, b, c\rangle, \langle d, e, f\rangle\}$. We'll refer to the individual clusters with one-based indexes: R_1, R_2 and S_1, S_2.

The algorithm begins with a loop over all clusters in R. Lines 4-8 set up the loop, which builds up a mapping M from each record to the cluster in R it is a member of. To save space, we will not show the entire contents of M, but as examples, $M[a] = 1$ and $M[b] = 2$. The loop also computes an array $Rsizes$ that stores the size of each cluster in R. The $Rsizes$ array will be updated over the course of the algorithm as we split pieces off of each cluster in R. In our example, the $Rsizes$ array will have the value 3 for both entries.

The algorithm then continues compute the cost of building each cluster $S_i \in S$. The first step is determining which clusters in R contain the records in S_i. Lines 14-21 build a structure $pMap$ that, for each cluster R_j in R, keeps a count of the records in S_i that are in R_j. If $R_j \cap S_i = \varnothing$, then there will be no entry in $pMap$ for S_j. Therefore there is at most one entry in $pMap$ for each record in S_i. In our example, the $pMap$ generated for S_1 will have $pMap[1] = 2$ and $pMap[2] = 1$, since the two records a, c are in R_1, whereas the one remaining record b comes from R_2. When

$pMap$ is generated for S_2, it will have $pMap[1] = 1$ and $pMap[2] = 2$.

To build S_i from the clusters in R, we must first split off the parts of the clusters in R that have the records in S_i. We can perform this splitting with a single split operation for each cluster that intersects S_i. Once those k pieces are split off, then we can merge them all together with $k-1$ merge operations. Lines 24-37 compute the cost for this series of operations, consulting $pMap$ to find out how many records must be split off of each cluster in R. In our example, the cost to construct S_1 would be computed as follows. First, $pMap[1]$ is 2, so we would have to split two records off of R_1. Since R_1 currently has size 3 (according to $Rsizes$), the cost for this split would be $f_s(2, 3-2)$. In the next iteration, we would consider $pMap[2] = 1$, and split 1 record away from R_2. This split would cost $f_s(1, 3-1)$. Now that there are two "fragments", we compute the cost to merge them together: $f_m(2, 1)$. This would end the loop and the cost for constructing S_1 would be $2 \times f_s(2, 1) + f_m(2, 1)$.

We note that on line 31, we update the $Rsizes$ array to reflect the fact that records have been split off of the clusters in R. After computing the cost to construct S_1, $Rsizes$ will have been updated to the sizes of the clusters in R without records in S_1. Specifically, $Rsizes[1] = 3 - 2 = 1$ and $Rsizes[2] = 3 - 1 = 2$.

The final details of the algorithm are to simply sum up the costs to construct all the S_i clusters, which is the merge distance from R to S.

D. EXPERIMENTS

D.1 Data

In the body of the paper, we use both synthetic and real data to evaluate various measures. Synthetic data lets us study many scenarios and understand when each measure is advantageous. Real data, on the other hand, provides a "sanity check" of the results found using synthetic data. In this section, we discuss how we generated the synthetic data and then describe the real datasets used.

Synthetic Data. We generate ER results that contain certain types of ER errors and have a given distribution of record sizes. Notice that our generator is different from ER benchmarks that produce inputs to ER algorithms. Instead, our generator produces possible *ER results*. An ER result R is generated in two steps. First, the gold standard G is generated using a given distribution of cluster sizes. Second, R is generated from G by adding ER errors. The possible types of errors in the ER results are categorized below. Notice that the errors are not necessarily distinct from each other (i.e., having one type of error may result in also having another type of error).

- Broken Entity: A cluster is split into two clusters of random sizes. For example, the record in the gold standard $G = \{\langle a, b, c, d, e\rangle\}$ splits into $\langle a, b\rangle$ and $\langle c, d, e\rangle$, resulting in $R = \{\langle a, b\rangle, \langle c, d, e\rangle\}$.
- Glued Entity: Two clusters are merged into a single cluster. For example, the two records in $G = \{\langle a, b, c\rangle, \langle d, e\rangle\}$ merge, resulting in $R = \{\langle a, b, c, d, e\rangle\}$.
- Misplaced Entity: A record within a cluster is detached and combined with another cluster. For example, the record c in the gold standard $G = \{\langle a, b, c\rangle, \langle d, e\rangle\}$ detaches from $\langle a, b, c\rangle$ and combines with $\langle d, e\rangle$, resulting in $R = \{\langle a, b\rangle, \langle c, d, e\rangle\}$.

Table 3 shows the parameters used to generate the ER results that contain the various types of ER errors above. We first generate a random gold standard G based on a given number of entities E and a cluster size distribution. We assume a Zipfian distribution with an exponent number e for the possible cluster sizes within the range $[1, C]$ where C is the maximum possible cluster size. The probability of a cluster having size k is thus $\frac{1/k^e}{\Sigma_{c=1}^{C}(1/c^e)}$. Once the gold standard G is generated, an ER result R is produced based on the gold standard. To generate broken entity errors, we split each cluster in G into two clusters of random sizes with probability b. To generate glued entities, we glue each pair of entities with a probability of g. We perform a transitive closure for all glued entities at the end. Finally, to generate misplaced entities, we remove a single record from a random cluster in G containing more than one record and attach it to a different random cluster. We also avoid misplacing a record that has already been misplaced before.

Table 3: Parameters for ER Result Generation

Parameter	Description	Value(s)
E	Number of entities	[10K,160K]
e	Zipfian exponent for cluster size distribution in G	1.5
C	Maximum cluster size	20
b	Probability of a cluster broken	[0.1,1.0]
g	Probability of two clusters glued together	[1e-5,1e-4]

Real Data. We used two real datasets. We used a comparison shopping dataset provided by Yahoo! Shopping, which 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 5,000 records that had the string "iPod" in their titles. 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 users. Again, we experimented on a random subset of size 5,000 of the hotel data. While we had a manually resolved gold standard for the hotel dataset, we did not have a gold standard for the shopping dataset and thus created one by running a pairwise comparison between all pairs of records using a strict matching criteria and then performing a transitive closure at the end.

We ran two ER algorithms on the hotel and shopping datasets to produce ER results. The Swoosh algorithm [9] uses a Boolean pairwise match function to compare records and a pairwise merge function to merge two records that match into a composite record. Swoosh starts comparing records in pairs and merges those that match. The merged records are compared again with other records for new iterative matches. The matching and merging repeats until no records match with each other. The match function for the shopping data compares the title, price, and category values of two records. For the hotel dataset, we compared the names and addresses of hotels. We also used an ER algorithm by Monge and Elkan [16] (ME) where records are sorted using an application-specific key and then clustered with a sequential scan. During the scan, each record is compared with the "representative" records of clusters and added to its closest cluster.

D.2 Conflicts in Real Datasets

In the body of the paper, we used synthetic datasets to show that conflicts can occur frequently among measures. In this section, we show that conflicts can also occur in real datasets as shown in Table 4, which shows the measure results for the two ER algorithms run on the shopping and hotel datasets. We added the distance results for BMD and accuracy results for the other measures. It is important to understand that for BMD, a *smaller* distance indicates a more accurate ER result. For the hotel dataset, the Swoosh algorithm performs better than the ME algorithm according to the BMD measure (again, the algorithm with the smaller distance is better), but not for pF_1 (higher accuracy is better). Hence, the pF_1 and BMD measures conflict. The pF_1 measure also conflicts with cF_1, which considers the Swoosh result more accurate. The other K and ccF_1 measures do not conflict with other measures because of the similar accuracies given to the two ER results (recall we set $\epsilon = 0.01$). For the shopping dataset, both pF_1 and BMD consider the Swoosh algorithm to be better than the ME algorithm while the other measures give similarly high accuracy values to both algorithms. We do not find any conflicts in this case. The results show that conflicts can indeed occur in real world applications, and evaluations of ER algorithms need to consider multiple measures (something that to date is seldom done).

Table 4: Measure results for real-world algorithms and datasets

	BMD	pF_1	cF_1	K	ccF_1
	Hotel				
Swoosh	427	0.34	0.88	0.95	0.93
ME	435	0.61	0.86	0.96	0.93
	Shopping				
Swoosh	29	0.86	0.98	0.98	0.99
ME	34	0.73	0.98	0.97	0.99

D.3 Comparing ER Algorithms with Configured Measures

In the main body of the paper, we discussed how the cost functions can be chosen based on application-specific knowledge. Table 5 shows how the Swoosh and ME algorithms can be compared using configured GMD measures. The ER results are identical to the ones used for Table 4. Each GMD measure gives certain information on how the two algorithms performed. For example, using the results of GMD_P and GMD_R, we know that the Swoosh algorithm is superior to the ME algorithm in terms of broken entity errors, but inferior in terms of glued entity errors, for both of the datasets. Comparing the results of BMD and GMD_H, we suspect that the Swoosh algorithm does a poor job in resolving large clusters because Swoosh has much higher GMD_H distances than those of ME, but similar BMD distances. (Recall that GMD_H is more sensitive to errors in large clusters than BMD.)

Table 5: Configured GMD results for real-world algorithms and datasets

	BMD	GMD_P	GMD_R	GMD_H	GMD_V
	Hotel				
Swoosh	427	2087	158	2672	0.177
ME	435	79	374	888	0.122
	Shopping				
Swoosh	29	28118	0	28147	0.084
ME	34	0	12794	12828	0.078

D.4 Runtime Performance

In the main body of the paper, we proposed an efficient algorithm for computing GMD measures. In this section, we compare the computation times for the BMD, pF_1, cF_1, K, ccF_1, and VI measures. (We omit the other configured GMD measures because their runtimes are similar to that of BMD.) For BMD, we used the Slice algorithm. For pF_1, we used two implementations: one uses the Slice algorithm while the other is a straightforward implementation that iterates through all record pairs of the ER result and the gold standard. Similarly for VI, we used an implementation using Slice (i.e., GMD_V) and a straightforward implementation that iterates through all pairs of clusters between the ER result and the gold standard. We implemented cF_1, ccF_1, and K in a straightforward way (as described in Section 5) because there are no better published algorithms. As a result, cF_1 was implemented with a linear time algorithm while ccF_1 and K were implemented with quadratic time algorithms. All the algorithms were implemented in Java, and our experiments were run in memory on a 2.4GHz Intel(R) Core 2 processor with 4GB of RAM.

Figure 6 shows the runtime plots for the measures. We experimented on 10K to 160K entities with the Zipfian exponent $e = 1.5$, and each ER result R had $\frac{|R|}{10}$ misplaced entities. Any implementation using the Slice algorithm is scalable to large ER results, with a runtime increasing linearly by the number of entities. Although the straightforward implementation of pF_1 is worst-case quadratic, in this experiment it shows linear average behavior (because clusters are small — average size is 3.5 — making the number of records to iterate over small). The straightforward implementation of VI is expensive even for a small number of entities, highlighting the runtime improvements when using Slice. The cF_1 algorithm is efficient and shows a linear increase in runtime. Finally, the runtimes of the K and ccF_1 algorithms grow quadratically against the number of entities and show the worst runtimes.

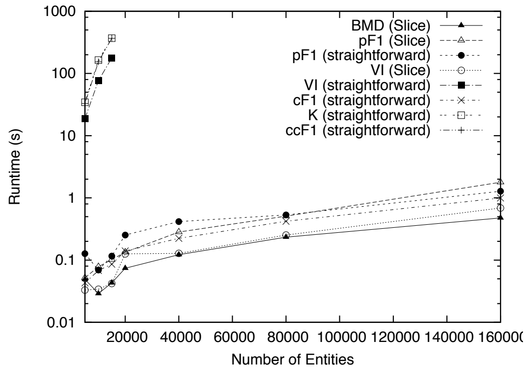


Figure 6: Scalability