Evaluating Entity Resolution Results (Extended version)

David Menestrina Steven Euijong Whang
Hector Garcia-Molina
Stanford University
{dmenest, swhang, hector}@cs.stanford.edu

November 6, 2009

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 extensive survey 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 to adjust two important parameters: sensitivity to error type and sensitivity to cluster size. This flexibility enables us to clearly understand the characteristics of a defined GMD measure. Surprisingly, a state-of-the-art clustering measure called Variation of Information is also a special case of our 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. As a result, both Variation of Information and pairwise F_1 can be computed in linear time.

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.

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 various ER algorithms [1, 2, 3, 4], but there is currently no agreed standard measure for evaluating ER results. Most works tend to use one ER measure

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

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 base record pairs that match, R_1 could be a better solution because it found 15 correct pairs (i.e., all base 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 base records while R_3 has found some non-matching base 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 base record pairs (unlike R_3).

Surprisingly, such conflicts between ER measures can occur frequently. Section 8.2 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 [5, 6]. 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. Surprisingly, at least two state-of-the-art measures are closely related to generalized merge distance: the Variation of Information (VI) [7] 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 the Slice algorithm) that efficiently computes generalized merge distance for a large class of cost functions that satisfy reasonable properties. As we argue in this paper (Section 6) 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∼4).

- 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 5∼6).
- 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 the GMD measure can be configured on two important parameters: sensitivity to error type and sensitivity to cluster size. Finally, we demonstrate the scalability of the Slice algorithm (Section 8).

2 Entity Resolution Model

In Entity Resolution (ER), our task is to determine:

1. the number of real-world entities in the input data, and
2. which pieces of information in the input data refer to the same real-world entity.

We define the problem formally as follows.

An ER problem consists of a set I of N input records. The result of an ER algorithm is a *partition* of I. A partition of a set S is a set of sets $\{s_1, s_2, s_3, \dots, s_n\}$ such that:

1. $\bigcup_i s_i = S$,
2. for all pairs s_i, s_j, $s_i \cap s_j = \varnothing$, and
3. for all sets s_i, $s_i \neq \varnothing$.

We refer to each s_i as a cluster. Although clusters are sets, we will denote them using $\langle , \rangle$ to distinguish them from sets that are not to be understood as members of a partition. For shorthand, we may in some cases represent a cluster 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.

Many ER algorithms [8, 9] return only a set of records, rather than a partition of the base records. They use a merge function to condense multiple records into a single record in the result. As an example, the result of resolving $r_1 = \{name$: J. Smith $\}$ and $r_2 = \{name$: John S. $\}$ might be the merge of the two records, $r_3 = \{name$: John Smith $\}$. Results that are not partitions are not directly covered by our model, but if we instrument any such algorithm to keep track of the lineage of records in the result, then we can use the lineage information to transform the result into a partition of the base records. In our example, this would mean remembering that r_3 was the result of merging the base records r_1 and r_2, and thus we would consider those two base records as members of the same cluster.

3 Evaluating ER Results

To evaluate an ER algorithm, we must compare its results to a gold standard. We define the gold standard S as a partition of the input records I. So we would like to assign a number to characterize how far away a result R is from the gold standard S. Let $g(R, S)$ be the function that performs this task of assigning a number.

We would first like to discuss a few properties that g may have. First, for g to be useful, we would like to be able to compare the values $g(R_1, S)$ with $g(R_2, S)$ to determine whether R_1 or R_2 is closer to S. If $g(R, S)$ generally increases as R and S get closer to each other, then we call g a similarity measure. If g generally decreases under the same circumstances, then we call g a distance measure. We note that if $g(x, y)$ is a similarity measure, then $g_d(x, y) = -g(x, y)$ is a distance measure.

Another property of $g(x, y)$ has to do with its range. If $g(x, y) \in [0, 1]$ for all x, y, then we say g is normalized. In some applications, it may be desirable to use a normalized similarity or distance measure. If g is not normalized, but has some other bounded range, it is trivial to normalize g to the range $[0, 1]$.

To provide some examples, Hamming distance and edit distance of strings are non-normalized distance measures. The Jaccard coefficient of sets is a normalized similarity measure.

We have so far avoided using the term "metric", as a metric is a formally defined mathematical concept. We now consider whether we should expect a distance measure g to be a metric. A proper metric must satisfy the following five properties: [10]

1. $g(x, y)$ is non-negative: $g(x, y) \geq 0$,
2. $g(x, y)$ satisfies the triangle inequality: $g(x, y) + g(y, z) \geq g(x, z)$,
3. $g(x, y)$ is symmetric: $g(x, y) = g(y, x)$,
4. $g(x, x) = 0$,
5. if $g(x, y) = 0$, then $x = y$.

We also define a similarity metric as a similarity measure $s(x, y)$ where the function $g(x, y) = c - s(x, y)$ is a metric for some value of c.

Of the five properties of metrics, we consider Properties 1 and 4 as reasonable enough to assume outright of any distance measure that we consider in this paper. Properties 2, 3, and 5, however, are subject to debate. Property 5 requires that the distance between distinct values be non-zero. However, when using common similarity measures such as precision or recall, it is certainly possible for two distinct values to have 100% similarity (zero distance). Property 2 does not have to hold where we only compare two sets R and S at a time (i.e., we do not measure the distance of a sequence of more than two ER results).

The question of symmetry (Property 3) is more interesting. Consider the following two cases:

1. $R = \{\langle a\rangle, \langle b\rangle\}$ and $S = \{\langle a, b\rangle\}$
2. $R = \{\langle a, b\rangle\}$ and $S = \{\langle a\rangle, \langle b\rangle\}$

In the first case, our ER algorithm has missed a match. In the second, it has found a match where there should not have been one. If false negatives and false positives are considered equally bad, then the two cases have equal distance and our similarity or distance measures may be symmetric. However, in many cases, we may wish to consider measures that have different penalties for different types of errors. So in the most general case, symmetry may not be a property of a method of evaluating the results of ER.

Functions that satisfies all properties of a metric except for Properties 2, 3, or 5 are called semimetrics, quasimetrics, or pseudometrics, respectively [10]. Since the measures we consider in this paper may not satisfy the three properties above, they may be referred to as premetrics [10]. In this paper we will avoid the use of the term "metric" altogether and use the more general term "measure".

4 Existing Measures

We review state-of-the-art measures for evaluating ER results and motivate our edit distance measure. 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 [11]. Hence, the ER literature has historically only adopted a small subset of all clustering measures for IR.

4.1 Pairwise Comparison

The pairwise comparison approach counts the number of pairs of base 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 [12, 13, 14, 15] use pairwise precision and pairwise recall to evaluate ER results while earlier works [16, 17, 18] 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 [15, 19] use a variant of pairwise recall while taking into account the reduced number of record comparisons due to blocking techniques. Another work [20] 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, 21, 22, 23, 24, 25, 26, 11, 27, 28, 29, 2, 30] 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%.

4.2 Cluster-level Comparison

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

cF_1 The cluster F_1 measure [31, 28, 29, 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 base 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 [32, 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 base records. (Notice that the records r and s are considered as sets of base 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 [8] 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 [20] uses a variant of ccF_1 that uses a different similarity equation and gives weights to the coefficients when adding them.

4.3 Basic Merge Distance

Edit distance is a common measure in other domains such as string-to-string matching [5, 6] where the basic operations are inserts, deletes, updates, and swaps. In the ER domain (as in clustering), there are fundamental "edit" operations such as cluster splits and merges [33] that are frequently used to resolve records.

A measure based on splits and merges was first proposed by Al-Kamha et al. [34], which we call basic merge distance. Since basic merge distance will be the basis for our generalized merge distance measure, we will describe basic merge distance in more detail than the other measures we have covered.

The basic merge distance (BMD) is defined as the minimum number of cluster merges and splits required to modify an ER result R into another result S. (In most cases, we will have $S = G$, the gold standard.) In the example from Table 1, only one merge is required to get from R_2 to G (i.e., BMD = 1). Result R_1 is comparatively further away from G, as BMD = 2.

In addition, 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 have a BMD of 2, 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 $\langle c, d\rangle$ (resulting in a BMD of 4). 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.

We now formalize the definition of BMD.

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: $P : o_1, o_2$.

Definition 4.3. *A path from partition R to partition R' is a sequence of operations $o_1, o_2, \ldots, o_n$ where $R' = R : o_1, o_2, \ldots, o_n$ 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 R' if for any operation that is a merge $o_1 = c_1, c_2 \to c$, then there exists a cluster $p \in R'$ where $c \subseteq p$.*

Definition 4.4. *The BMD from a partition R to a partition S is the number of operations in the shortest legal path from R to S.*

While the BMD measure also operates at the cluster level, we categorize it separately from cluster-level comparison approaches because clusters are dynamically edited using basic operations instead of being statically compared.

4.4 Variation of Information

Another closely related work to merge distance measure is a state-of-the-art clustering measure called Variation of Information [7] (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.

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

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

5 Generalized Merge Distance

The definition of basic merge distance (Section 4.3) 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. In Section 5.3 we show that this space includes distance measures closely related to the pairwise precision and recall measures of Section 4.1, as well as the VI measure of Section 4.4.

Definition 5.1. *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 $x, y \to z$ is $f_m(|x|, |y|)$, and*
- *the cost of a split operation $z \to x, y$ is $f_s(|x|, |y|)$.*

Clearly, the BMD measure described in Section 4.3 is the same as the GMD measure when $f_m(x, y) = f_s(x, y) = 1$.

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

1. Operations cannot have negative cost: $f_m(x,y) \geq 0$ 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.

We begin by noting that merge and split operations are often commutative. That is, $R : o_1, o_2 = R : o_2, o_1$ for many pairs of operations. In fact, operations are always commutative except when the input of one operation is the output of another.

As an example, consider the path defined by operations o_1, o_2, o_3, o_4, o_5 in Table 2. Operations o_1 and o_3 can be executed in either order, as their inputs and outputs do not overlap. Operation o_4, on the other hand, takes the output of o_3 and one of the outputs of o_2 as input. Therefore, o_4 must be performed after both o_2 and o_3.

Table 2: Example path

Operation	Result
—	$\langle abc\rangle, \langle de\rangle, \langle f\rangle, \langle g\rangle$
o_1: $\langle abc\rangle, \langle de\rangle \rightarrow \langle abcde\rangle$	$\langle abcde\rangle, \langle f\rangle, \langle g\rangle$
o_2: $\langle abcde\rangle \rightarrow \langle ae\rangle, \langle bcd\rangle$	$\langle ae\rangle, \langle bcd\rangle, \langle f\rangle, \langle g\rangle$
o_3: $\langle f\rangle, \langle g\rangle \rightarrow \langle fg\rangle$	$\langle ae\rangle, \langle bcd\rangle, \langle fg\rangle$
o_4: $\langle fg\rangle, \langle bcd\rangle \rightarrow \langle bcdfg\rangle$	$\langle ae\rangle, \langle bcdfg\rangle$
o_5: $\langle ae\rangle, \langle bcdfg\rangle \rightarrow \langle abcdefg\rangle$	$\langle abcdefg\rangle$

The commutativity of operations in a path can be represented with a precedence graph.

Definition 5.2. *The precedence graph G for a path $o_1, o_2, \ldots, o_n$ is a directed graph that specifies the order in which the operations must take place. We build the graph by creating a vertex for each operation. For each pair of operations o_i, o_j with $i < j$, if an output cluster p of o_i is an input of o_j and there is no k between i and j where o_k also has p as an input, then we add an edge from o_i to o_j.*

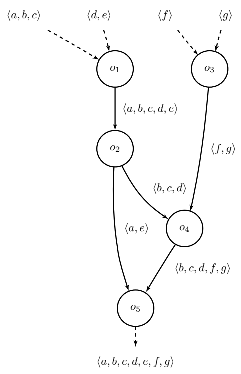


Figure 1: A precedence graph.

Figure 1 shows the precedence graph for the path from our example. For clarity, the edges are labeled with the cluster that creates the dependence between two operations. Dashed lines in this figure are not part of the precedence graph; they are present only to show the inputs and outputs of operations that would not be apparent from the precedence graph alone. The graph clearly illustrates that o_3 can commute with o_1 and o_2, but no other pairs of operations are commutable. To construct a path with all splits preceding all merges, we can often simply use commutativity to move the split operations to the front. However, if there is an edge from a merge to a split in the precedence graph, commutativity will not help. In that case, we must invoke a more complex transformation which we call a "merge-split swap".

We will first illustrate a merge-split swap with an example, and then formally define the transformation. In our running example, merge operation o_1 has an edge to split operation o_2 (which we would like to move "up" to the beginning of the path). We will "swap" the order of the operations by replacing o_1 with a split operation o'_1 with an edge to a merge operation o'_2 that replaces o_2. The input of o'_1 will be $\langle a,b,c\rangle$, the larger of the two inputs of o_1. We then split that input into two clusters, one of which has the same size as the smaller of the outputs of o_2. We will arbitrarily choose $\langle a,b\rangle$ as the output of size 2, and thus $o'_1 = \langle a,b,c\rangle \rightarrow \langle a,b\rangle, \langle c\rangle$. We will define o'_2 as the merge of the "leftover" cluster $\langle c\rangle$ with the smaller of the inputs to o_1. So $o'_2 = \langle c\rangle, \langle d,e\rangle \rightarrow \langle c,d,e\rangle$.

Note that the sequences o_1, o_2 and o'_1, o'_2 do not produce the same result (the former yields $\langle a,e\rangle, \langle b,c,d\rangle$ and the latter yields $\langle a,b\rangle, \langle c,d,e\rangle$). However, the two sequences each produce 2 clusters of the same size. To compensate for the different output, we modify operations "downstream" from o'_1, o'_2 so they work with the new clusters, as shown in Figure 2.

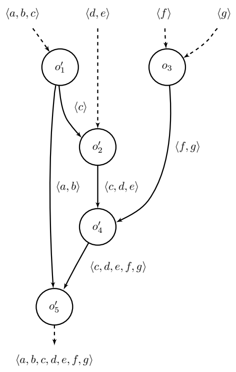


Figure 2: Precedence graph for the transformed path.

We note that the result of the transformed path is the same as the result of the original path: $\{\langle a,b,c,d,e,f,g\rangle\}$. But suppose the original path only had operations $o_1, \ldots, o_4$. Then the result before the transformation would be $\{\langle a,e\rangle, \langle b,c,d,f,g\rangle\}$, and the result of the transformed path would be $\{\langle a,b\rangle, \langle c,d,e,f,g\rangle\}$—a different result! As it turns out, as long as the original path was a legal path, the result after the transformation will be the same. Removing o_5 from the path changes the result in such a way that o_1 becomes an invalid merge ($\langle a,b,c,d,e\rangle$ is not a subset of any cluster in the result). Lemma 5.1 will prove that the merge-split swap transformation can be applied to any legal path without changing the result of the path.

Figure 3 provides a generalized depiction of the merge-split swap transformation. It will be useful as a reference for the following formal definition.

Definition 5.3. *The merge-split swap transformation is defined as follows. Consider a path with a merge operation $m = \pi_1, \pi_2 \rightarrow \pi_{12}$ with an edge to a split operation $s = \pi_{12} \rightarrow \pi_3, \pi_4$ in the precedence graph. Let $|\pi_1| \geq |\pi_2|$ and $|\pi_3| \geq |\pi_4|$.*

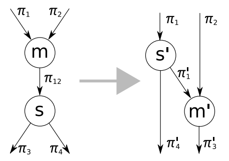


Figure 3: A merge with an edge to a split is rewritten into a split with an edge to a merge.

We construct two new operations: $s' = \pi_1 \rightarrow \pi'_1, \pi'_4$ and $m' = \pi'_1, \pi_2 \rightarrow \pi'_3$. When constructing these operations, we choose a $\pi'_4 \subseteq \pi_1$ where $|\pi'_4| = |\pi_4|$. (It is possible in this scenario that $|\pi'_1| = 0$, which would make s' and m' illegal operations. However, in this case we can consider s' and m' to be zero cost no-ops) The transformation begins by replacing operation m with s', and replacing s with m'.

The remaining changes to the path simply fix subsequent operations to use π'_3 and π'_4 as input, instead of π_3 and π_4. Defining this process formally, we construct a one-to-one mapping M of records. We will employ an extended notation that allows us to apply M to sets and operations, e.g. $M(S) = \{M(x) : x \in S\}$ and $M(\pi_1, \pi_2 \rightarrow \pi_{12}) = M(\pi_1), M(\pi_2) \rightarrow M(\pi_{12})$. For all records $r \notin \pi_3 \cup \pi_4$, we define $M(r) = r$. For the records in π_3 and π_4, we define M such that $M(\pi_3) = \pi'_3$ and $M(\pi_4) = \pi'_4$.

The merge-split swap transformation is complete when we replace each subsequent operation o in the path with $M(o)$.

Definition 5.1. *Given a legal path p from R to S, applying a merge-swap split transformation results in a legal path p' from R to S with cost less than or equal to the cost of p.*

Proof. In this proof, we will reuse the terms introduced in the definition of the merge-split swap: clusters $\pi_1, \pi_2, \pi_3, \pi_4, \pi'_3, \pi'_4$, operations m, s, s', m', and mapping M.

We make three claims about p':

1. Path p' is a path from R to S.

2. Path p' is a *legal* path from R to S.

3. The cost of p' is less than or equal to the cost of p.

To prove Claim 1, we first name the operations in our paths to $o_1, \ldots, o_n$ and $o'_1, \ldots, o'_n$, respectively. Suppose that the split to be transformed is $o_k = s$, and therefore $o'_k = m'$. We note that $R : o'_1, \ldots, o'_k = M(R : o_1, \ldots, o_k)$. That is, after applying p' up through the kth operation, then we have the same result as applying the first k operations of p and then applying M to each set in the resulting clusters. For each subsequent operation o, $o' = M(o)$, so it is clear that $R : o'_1, \ldots, o'_{k+i} = M(R : o'_1, \ldots, o'_{k+i})$ for positive i. Therefore, $R : p' = M(R : p) = M(S)$.

Now, the operation m was a merge of π_1 and π_2. Since p is a legal path, $\pi_1 \cup \pi_2$ must be a subset of some cluster π_f in S. Since $\pi_1 \cup \pi_2 = \pi_3 \cup \pi_4$, we can write $\pi_f = \pi_3 \cup \pi_4 \cup \pi_{extra}$. Applying M to π_f, we get $\pi'_3 \cup \pi'_4 \cup \pi_{extra} = \pi_1 \cup \pi_2 \cup \pi_{extra}$. So $M(\pi_f) = \pi_f$. Since M is an identity function for all records not in π_1 or π_2, the other clusters in S are also unaffected by the application of M. Therefore, $M(S) = S$, which proves Claim 1.

Further, since records r and $M(r)$ are always in the same cluster in the result, if any merge operation o is a legal merge in p, then the corresponding operation $M(o)$ in p' must also be a legal merge. The only questionable merge remaining is m', which is legal since π'_3 is a subset of $\pi_f \in S$. Therefore, p' is a legal path, which proves Claim 2.

To prove Claim 3 we note that applying M to an operation o cannot change the size of the clusters involved, and therefore, the cost of $M(o)$ must be the same as the cost of o. So all corresponding operations in p and p' have equal cost, other than m, s, s' and m'. The only difference in cost can come from the cost difference between s and s', and m and m'. The combined cost for m and s is:

$$f_m(|\pi_1|, |\pi_2|) + f_s(|\pi_3|, |\pi_4|)$$

The combined cost for m' and s' is:

$$f_m(|\pi'_1|, |\pi_2|) + f_s(|\pi'_1|, |\pi'_4|)$$

Recall from Definition 5.3 that $|\pi_1| \geq |\pi_2|$, $|\pi_3| \geq |\pi_4|$, and $|\pi'_4| = |\pi_4|$. We also have that $|\pi_1| + |\pi_2| = |\pi_3| + |\pi_4| = |\pi'_3| + |\pi'_4|$ and $|\pi_1| = |\pi'_1| + |\pi'_4|$.

From these facts, we find that $|\pi'_1| \leq |\pi_1|$, $|\pi'_1| \leq |\pi_3|$, and of course $|\pi'_4| = |\pi_4|$. Comparing the cost expressions with these three facts, we see that all arguments to the f_m and f_s functions are lower or equal for operations m' and s' than they are for m and s. By the monotonicity property of the split and merge functions, the cost for m' and s' must be less than or equal to the cost of m and s. Therefore the cost of p' is less than or equal to the cost of p. This proves Claim 3 and therefore we have proven the lemma. □

Theorem 5.1. *For any partitions R and S, there exists a minimum cost legal path from R to S where the precedence graph for the path has no edge from any merge operation to a split operation.*

Proof. Consider a minimum cost legal path p from R to S. If the precedence graph of p has a merge with an edge to a split, we can apply a merge-split swap transformation in order to obtain a path that (according to Lemma 5.1) must also be a legal and minimum cost path from R to S. In fact, we can repeat the transformation until there no longer exists an edge from a merge operation to a split operation in the precedence graph of the path. □

Theorem 5.2. *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. This result follows directly from Theorem 5.1 and the rules of commutativity. Since there is no edge from a merge operation to a split operation in the precedence graph, the split operations may all be commuted to the beginning of the path. □

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, as if there were 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.

We also define the "split point" of a splits-first path.

Definition 5.4. *In a splits-first path from R to S, we can apply all of the split operations to R to get a partition we call the* split point. *We denote the split point of a path p applied to a partition R as $R : splits(p)$.*

5.1 Operation Order Independence

Definition 5.5. *A* fragment *of partitions R and S is any set of the form $s \cap r$ where $r \in R, s \in S, r \cap s \neq \varnothing$. We denote the set of all fragments of R and S as $R \sqcap S$. Since every record is in exactly one fragment, $R \sqcap S$ is a partition as well.*

As an example, suppose $R = \{\langle a,c,e\rangle, \langle b,d,f\rangle\}$ and $S = \{\langle a,b,c\rangle, \langle d,e,f\rangle\}$. In that case, $R \sqcap S = \{\langle a,c\rangle, \langle e\rangle, \langle b\rangle, \langle d,f\rangle\}$.

If a splits-first path fails to break R down to the fragments by the end of the split phase, then it cannot possibly result in S after the merge phase is complete. Therefore, the fragments of R and S can be seen as the minimum required splitting performed by a splits-first path from R to S.

Definition 5.2. *Given a splits-first path from R to S, any cluster in $R : splits(p)$ is a subset of some $f \in R \sqcap S$.*

Proof. Consider any cluster $\pi \in R : splits(p)$. Since $\pi \in R : splits(p)$, it is the result of a series of split operations on R. Therefore, $\pi \subseteq r$ for some $r \in R$. The remaining operations in the path are all merges, so $\pi \subseteq s$ for some cluster $s \in S$. Now we have that $\pi \subseteq r \cap s$, and $r \cap s \in R \sqcap S$. So any cluster in $R : splits(p)$ must be a subset of some $f \in R \sqcap S$. □

We now define a term for a path that performs this minimum required splitting:

Definition 5.6. *A* bare necessities path *from R to S is any splits-first path p from R to S where $R : splits(p) = R \sqcap S$.*

For many split and merge functions, there always exists a bare necessities path from R to S that is also a minimum cost path. In fact, if the functions satisfy a property we call "operation order independence", then any bare necessities path from R to S is in fact a minimum cost path.

Definition 5.7. *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 π_x, π_y, and π_z (with sizes x, y, and z, respectively) all together into a single cluster. If we merge π_x and π_y together first, and then merge the resulting cluster with π_z, observe that the resulting cost would be given by the left-hand side of the equation in Definition 5.7. The cost of merging π_x and π_z together first, and then merging π_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.

Now that we have defined operation order independence, we note two simple classes of functions that satisfy this property: $F(x,y) = k$ and $F(x,y) = kxy$. (One can easily verify the property holds for these classes by plugging them into the equation in Definition 5.7.) The first class includes the BMD measure of Section 4.3 and the second class of measures can be used to directly compute (see Section 5.3) the pairwise precision and recall measures of Section 4.1. We also note that functions of the form $F(x,y) = k_1 + k_2xy$ also satisfy this property, and these may provide an interesting "blend" of the two classes above. These are not the only functions that satisfy this property, and it turns out that the class of functions that satisfy the property (studied in [35]) is, in fact, quite vast.

The class of operation order independent functions has been studied in [35]. That paper proves the general form of an operation order independent function to be:

$$F(x,y) = B(x,y) + f(x+y) - f(x) - f(y)$$

In this formulation, $f(x)$ is any arbitrary function. On the other hand, $B(x,y)$ is a function that must satisfy many properties, including skew symmetry: $B(x,y) + B(y,x) = 0$. Since we assume the property of symmetry on our cost functions, we require that $B(x,y) = 0$. Therefore, the most general form for operation order independent functions is $f(x+y) - f(x) - f(y)$. We can verify that if $f(x) = -k$ then $F(x) = k$ and if $f(x) = \frac{k}{2}x^2$ then $F(x,y) = kxy$. This result shows that there is actually a large class of functions that satisfy operation order independence.

Definition 5.3. *Given clusters $\pi_1, \pi_2, \ldots, \pi_n$, if f_m is operation order independent, then all sequences of merges that result in $\bigcup \pi_i$ have equal cost.*

Proof. Consider the class of pairs P with the left element being a cluster and the right element being a numerical cost associated with that cluster. We map the clusters π_i to the corresponding pair $(\pi_i, 0)$, indicating that we can obtain any π_i with zero cost. We can now define an operator $\oplus$ on P that performs the merge of the clusters in two pairs, and computes the total cost necessary to obtain that cluster:

$$(p_1, x) \oplus (p_2, y) = (p_1 \cup p_2, x + y + f_m(|p_1|, |p_2|)$$

Due to the properties f_m has (symmetry and operation order independence), it is easy to show that $\oplus$ is both associative and commutative.

Any sequence of merges of the base clusters that results in $\bigcup \pi_i$ maps to an application of the $\oplus$ operator on all of the π_i clusters. Since $\oplus$ is commutative and associative, the order of merge operations does not affect the cost to generate the result. So all sequences of merges with that result have equal cost. $\square$

Definition 5.4. *If f_s is operation order independent, then all sequences of splits starting from a cluster π that result in $\pi_1, \pi_2, \ldots, \pi_n$ have equal cost.*

Proof. We note that any sequence of splits leading to the π_i clusters can be reversed to obtain a sequence of merges from the π_i clusters to π. If we let $f_m = f_s$, then the cost of the merge sequence is equal to the cost of the split sequence. By applying Lemma 5.3, we get that all such sequences have equal cost. $\square$

Theorem 5.3. *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. Take any minimum cost splits-first path p_{min} from R to S. (By Theorem 5.2 such a path always exists.) Now take any bare necessities path p from R to S. By Lemma 5.2, every cluster in $R : splits(p_{min})$ is a subset of some cluster in $R : splits(p)$ (which is $R \sqcap S$ by definition of a bare necessities path). Given this subset relationship, it is clear that we can append split operations to the splits phase of p such that we arrive at $R : splits(p_{min})$. Further, we can append merge operations to the end of the splits phase to merge these clusters back down to $R \sqcap S$. With this process, we can extend p to a new path p' from R to S where $R : splits(p') = R : splits(p_{min})$.

The new path p' contains all of the operations of p plus some extra operations in the middle. Therefore, the cost of p' is greater than or equal to the cost of p. However, by Lemma 5.3 and Lemma 5.4, the cost of p' must be equal to the cost of p_{min}, since $R : splits(p') = R : splits(p_{min})$. Path p_{min} is a minimum cost legal path from R to S, and p must have lower or equal cost. So the bare necessities path p is also a minimum cost legal path from R to S. $\square$

We now demonstrate that a bare necessities path may be easily constructed for any given partitions R and S. Let p_0 be the null path that performs no operations, so $R = R : p_0$. We define the splits in this path inductively: $p_{i+1} = p_i, \pi \to \pi_1, \pi_2$ where $\pi \in R : p_0$, and non-empty π_1 and π_2 are, respectively, $\pi \cap s$ and $\pi - s$ for some $s \in S$. We extend the path until all clusters in $R : p_i$ are subsets of clusters in S.

We then extend the path further with only merge operations: $\pi_1, \pi_2 \to \pi_1 \cup \pi_2$ where $\pi_1, \pi_2 \subseteq s$ for some $s \in S$. This completes a bare necessities path from R to S.

This construction suggests an algorithm for computing GMD when the merge and split functions are operation order independent functions, since it suffices to construct a bare necessities path and compute its cost. We will describe such an algorithm (called the Slice algorithm) in Section 6.

5.2 Merge Precision and Recall

Another idea inspired by Theorem 5.2 is that we can consider the costs of splitting and merging separately. When a cluster in the result must be split in a path to the gold standard, it is usually due to a false positive in the result.[1] When two clusters are merged, it is usually due to a false negative. Therefore, the total cost of the split operations is much like an inverse measure of precision, and the total cost of the merge operations is much like an inverse measure of recall.

[1]In some configurations, a minimum cost path may split clusters that are found together in the destination. Consider $R = \{\langle a, b, c\rangle, d\}$, $S = \{\langle a, b, c, d\rangle\}$, $f_s(x, y) = 0$, and $f_m = x^3 + y^3$. A minimum cost path will split $\langle a, b, c\rangle$ to avoid the large cost of a merge with a cluster of size 3.

We can further use Theorem 5.2 to normalize these measures. The cost of all the split operations can never be more than the cost of splitting all clusters in R down into individual base records. Likewise, the cost of all the merge operations can never be more than the cost of merging individual base records up into the clusters in S. Let $\perp$ represent a partition in which each base records is alone in its own partition. We can then normalize merge precision and recall using the factors $MD_{f_m,f_s}(R,\perp)$ and $MD_{f_m,f_s}(\perp,S)$, respectively.

Let $C_m(R,S)$ and $C_s(R,S)$ refer to the total cost of merges and splits (respectively) in a minimum cost path from R to S. We can then define merge precision and recall as follows:

Definition 5.8. *The f_m, f_s merge precision from R to S is defined according to the following formula:*

$$MP_{f_m,f_s}(R,S) = 1 - \frac{C_s(R,S)}{GMD_{f_m,f_s}(R,\perp)}$$

The f_m, f_s merge recall from R to S is defined according to this similar formula:

$$MR_{f_m,f_s}(R,S) = 1 - \frac{C_m(R,S)}{GMD_{f_m,f_s}(\perp,S)}$$

Note that we subtract the normalized distance from 1 to turn these measures into similarity measures, rather than distance measures.

With a definition of precision and recall from the merge perspective, it is possible to use the standard methods (e.g., F_1) to combine the two into a single number.

5.3 Relationship to Other Measures

Several other measures are closely related to GMD. First, the BMD measure in [34] is exactly our GMD measure when $f_m(x,y) = f_s(x,y) = 1$. Second, the VI measure (Section 4.4) is a special case of GMD where f_m and f_s are chosen as follows:

Theorem 5.4. *$VI(R,S) = GMD(R,S)$ when $f_m(x,y) = f_s(x,y) = h(x+y) - h(x) - h(y)$, with $h(x) = \frac{x}{N}\log\frac{x}{N}$.*

Proof. First, we note that f_m and f_s here are order operation independent functions, as they have the form shown in Section 5.1 to be the most general form of an order independent function. Theorem 5.3 therefore tells us that the GMD from R to S is the cost of any bare necessities path from R to S.

Construct a bare necessities path $p = o_1, o_2, \ldots, o_m$ with $R : p = S$. We will use the shorthand notation $R^{(i)}$ to refer to $R : o_1, \ldots, o_i$, with $R^{(0)} = R$ and $R^{(m)} = S$. We show by induction that $GMD_{f_m,f_s}(R, R^{(k)}) = VI(R, R^{(k)})$ for all $0 \leq k \leq m$. We note that $GMD_{f_m,f_s}(R,R^{(0)}) = VI(R,R^{(0)}) = 0$ which provides with a base case for our induction.

Now, assuming the inductive hypothesis, $GMD_{f_m,f_s}(R,R^{(i-1)}) = VI(R,R^{(i-1)})$, we will show that $GMD_{f_m,f_s}(R,R^{(i)}) = VI(R,R^{(i)})$. Let us evaluate the change in VI when we perform operation o_i.

$$\begin{aligned}
\Delta VI &= VI(R,R^{(i)}) - VI(R,R^{(i-1)}) \\
&= H(R) + H(R^{(i)}) - 2I(R,R^{(i)}) \\
&\quad - H(R) - H(R^{(i-1)}) + 2I(R,R^{(i-1)}) \\
&= H(R^{(i)}) - H(R^{(i-1)}) \\
&\quad - 2(I(R,R^{(i)}) - I(R,R^{(i-1)})) \\
&= \Delta H - 2\Delta I
\end{aligned}$$

In the above equation, we have introduced $\Delta H = H(R^{(i)}) - H(R^{(i-1)})$ and $\Delta I = I(R,R^{(i)}) - I(R,R^{(i-1)})$. The value of ΔH is the change in entropy created by performing the operation, and ΔI is the change in information shared with R.

We must handle the case of a split separately from the case of a merge. So for the time being, suppose that o_i is a split: $o_i = \pi \rightarrow \pi_1, \pi_2$. First, let us consider ΔH:

$$\begin{aligned}
\Delta H &= H(R^{(i)}) - H(R^{(i-1)}) \\
&= -\sum_{r \in R^{(i)}} h(|r|) + \sum_{r \in R^{(i-1)}} h(|r|) \\
&= -h(|\pi_1|) - h(|\pi_2|) + h(|\pi|) \\
&= h(|\pi|) - h(|\pi_1|) - h(|\pi_2|)
\end{aligned}$$

Now, we consider ΔI:

$$\begin{aligned}\Delta I &= I(R, R^{(i)}) - I(R, R^{(i-1)})\\ &= \sum_{r \in R} \sum_{s \in R^{(i)}} \frac{|r \cap s|}{N} \log \frac{|r \cap s| \times N}{|r| \times |s|}\\ &\quad - \sum_{r \in R} \sum_{s \in R^{(i-1)}} \frac{|r \cap s|}{N} \log \frac{|r \cap s| \times N}{|r| \times |s|}\end{aligned}$$

The only difference between these two summations is in the terms involving π for the first sum, and π_1, π_2 in the second sum. Since o_i is a split in a splits-first path, it is preceded only by splits, and therefore π, π_1, π_2 are all subsets of some single cluster $r^* \in R$. Therefore, for any $r \in R$ where $r \neq r^*$, $r \cap \pi = \varnothing$, and the same is true of π_1 and π_2. This fact brings ΔI down to the following:

$$\begin{aligned}\Delta I &= \frac{|r^* \cap \pi|}{N} \log \frac{|r^* \cap \pi| \times N}{|r^*| \times |\pi|}\\ &\quad - \frac{|r^* \cap \pi_1|}{N} \log \frac{|r^* \cap \pi_1| \times N}{|r^*| \times |\pi_1|}\\ &\quad - \frac{|r^* \cap \pi_2|}{N} \log \frac{|r^* \cap \pi_2| \times N}{|r^*| \times |\pi_2|}\\ &= \frac{|\pi|}{N} \log \frac{|\pi| \times N}{|r^*| \times |\pi|} - \frac{|\pi_1|}{N} \log \frac{|\pi_1| \times N}{|r^*| \times |\pi_1|}\\ &\quad - \frac{|\pi_2|}{N} \log \frac{|\pi_2| \times N}{|r^*| \times |\pi_2|}\\ &= \frac{|\pi|}{N} \log \frac{N}{|r^*|} - \frac{|\pi_1|}{N} \log \frac{N}{|r^*|} - \frac{|\pi_2|}{N} \log \frac{N}{|r^*|}\\ &= \frac{|\pi| - |\pi_1| - |\pi_2|}{N} \log \frac{N}{|r^*|}\\ &= \frac{0}{N} \log \frac{N}{|r^*|}\\ &= 0\end{aligned}$$

Now, we continue to compute ΔVI.

$$\begin{aligned}\Delta VI &= \Delta H - 2\Delta I\\ &= h(|\pi|) - h(|\pi_1|) - h(|\pi_2|) - 2 \times 0\\ &= h(|\pi_1| + |\pi_2|) - h(|\pi_1|) - h(|\pi_2|)\\ &= f_s(|\pi_1|, |\pi_2|)\end{aligned}$$

So the cost of the operation o_i is exactly ΔVI, which allows us to prove the inductive step in the case that o_i is a split:

$$\begin{aligned}VI(R, R^{(i)}) &= VI(R, R^{(i-1)}) + \Delta VI\\ &= GMD_{f_m, f_s}(R, R^{(i-1)}) + f_s(|\pi_1|, |\pi_2|)\\ &= GMD_{f_m, f_s}(R, R^{(i)})\end{aligned}$$

Now, we need to repeat this procedure assuming that o_i is a merge: $o_i = \pi_1, \pi_2 \rightarrow \pi$. Starting with computing ΔH:

$$\begin{aligned}\Delta H &= H(R^{(i)}) - H(R^{(i-1)})\\ &= -\sum_{r \in R^{(i)}} h(|r|) + \sum_{r \in R^{(i-1)}} h(|r|)\\ &= h(|\pi_1|) + h(|\pi_2|) - h(|\pi|)\end{aligned}$$

Note that ΔH in the merge case is the opposite of ΔH in the split case.

Now, we proceed to compute ΔI. The merge case is trickier, though, as π_1 and π_2 may have components of many clusters of R. Since the path is a bare necessities path, we can let $\pi_1 = \bigcup_{f \in F} f$ for some set of fragments $F \subset R \sqcap S$. We will use the notation f_j to refer to an individual element of F. Similarly, let $\pi_2 = \bigcup_{g \in G} g$, with $G \subset R \sqcap S$, and g_j referring to an individual element of G.

Since $F, G \subset R \sqcap S$, each element of these sets is a subset of some cluster in R. That is, for each f_j, there is a corresponding $c_j \in R$ where $f_j \subseteq c_j$. Likewise, for each g_j, there is a corresponding $d_j \in R$ where $g_j \subseteq d_j$. We will continue to use c_j and d_j to refer to the clusters in R that f_j and g_j came from, respectively. We note that $\pi_1 \cup \pi_2 = \varnothing$, so

We begin with the ΔVI equation from earlier, and simplify it by leaving only the terms that use the clusters involved in the merge operation:

$$\begin{aligned}
\Delta I &= \sum_{r\in R}\sum_{s\in R^{(i)}} \frac{|r\cap s|}{N}\log\frac{|r\cap s|\times N}{|r|\times|s|} \\
&\quad -\sum_{r\in R}\sum_{s\in R^{(i-1)}} \frac{|r\cap s|}{N}\log\frac{|r\cap s|\times N}{|r|\times|s|} \\
&= \sum_j \frac{|c_j\cap\pi|}{N}\log\frac{|c_j\cap\pi|\times N}{|c_j|\times|\pi|} \\
&\quad +\sum_j \frac{|d_j\cap\pi|}{N}\log\frac{|d_j\cap\pi|\times N}{|d_j|\times|\pi|} \\
&\quad -\sum_j \frac{|c_j\cap\pi_1|}{N}\log\frac{|c_j\cap\pi_1|\times N}{|c_j|\times|\pi_1|} \\
&\quad -\sum_j \frac{|d_j\cap\pi_2|}{N}\log\frac{|d_j\cap\pi_2|\times N}{|d_j|\times|\pi_2|} \\
&= \sum_j \left[\frac{|f_j|}{N}\log\frac{|f_j|\times N}{|c_j|\times|\pi|}+\frac{|g_j|}{N}\log\frac{|g_j|\times N}{|d_j|\times|\pi|}\right] \\
&\quad -\sum_j \left[\frac{|f_j|}{N}\log\frac{|f_j|\times N}{|c_j|\times|\pi_1|}+\frac{|g_j|}{N}\log\frac{|g_j|\times N}{|d_j|\times|\pi_2|}\right] \\
&= \sum_j \frac{|f_j|}{N}\left(\log\frac{|f_j|\times N}{|c_j|\times|\pi|}-\log\frac{|f_j|\times N}{|c_j|\times|\pi_1|}\right) \\
&\quad +\sum_j \frac{|g_j|}{N}\left(\log\frac{|g_j|\times N}{|d_j|\times|\pi|}-\log\frac{|g_j|\times N}{|d_j|\times|\pi_2|}\right) \\
&= \sum_j \left[\frac{|f_j|}{N}\log\frac{|\pi_1|}{|\pi|}+\frac{|g_j|}{N}\log\frac{|\pi_2|}{|\pi|}\right] \\
&= \frac{|\pi_1|}{N}\log\frac{|\pi_1|}{|\pi|}+\frac{|\pi_2|}{N}\log\frac{|\pi_2|}{|\pi|} \\
&= \frac{|\pi_1|}{N}\log|\pi_1|+\frac{|\pi_2|}{N}\log|\pi_2|-\frac{|\pi_1|+|\pi_2|}{N}\log|\pi| \\
&= \frac{|\pi_1|}{N}\log\frac{|\pi_1|}{N}+\frac{|\pi_2|}{N}\log\frac{|\pi_2|}{N}-\frac{|\pi|}{N}\log\frac{|\pi|}{N} \\
&= h(|\pi_1|)+h(|\pi_2|)-h(|\pi|)
\end{aligned}$$

Again, we continue to compute ΔVI.

$$\begin{aligned}
\Delta VI &= \Delta H - 2\Delta I \\
&= h(|\pi_1|)+h(|\pi_2|)-h(|\pi|) \\
&\quad -2(h(|\pi_1|)+h(|\pi_2|)-h(|\pi|)) \\
&= h(|\pi|)-h(|\pi_1|)-h(|\pi_2|) \\
&= h(|\pi_1|+|\pi_2|)-h(|\pi_1|)-h(|\pi_2|) \\
&= f_m(|\pi_1|,|\pi_2|)
\end{aligned}$$

So the cost of the operation o_i is exactly ΔVI, which allows us to prove the inductive step in the case that o_i is a merge:

$$\begin{aligned}
VI(R,R^{(i)}) &= VI(R,R^{(i-1)})+\Delta VI \\
&= GMD_{f_m,f_s}(R,R^{(i-1)})+f_m(|\pi_1|,|\pi_2|) \\
&= GMD_{f_m,f_s}(R,R^{(i)})
\end{aligned}$$

We have proven the inductive step in the case that o_i is either a merge or a split, and therefore we have proven the inductive step completely. Therefore, by induction, $GMD_{f_m,f_s}(R,R^{(k)}) = VI(R,R^{(k)})$ for all $0 \le k \le m$, and therefore $GMD_{f_m,f_s}(R,S) = VI(R,S)$. □

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 5.5. *$PairPrecision(R,S)$=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. We will prove this theorem for $f_m(x,y) = 0$ and $f_s(x,y) = xy$. The other case is symmetric.

A split operation $\pi \to \pi_1,\pi_2$ has a cost of $|\pi_1|\times|\pi_2|$. When we apply this operation to a partition, the result will be missing all pairs consisting of a record in π_1 and a record in π_2. There are $|\pi_1|\times|\pi_2|$ such pairs, so it turns out the cost of the split is the same as the reduction in the number of pairs.

Since f_m and f_s are operation order independent functions, Theorem 5.3 tells us that any bare necessities path from R to S has minimum cost. The splits in a bare necessities path will remove all pairs in $Pairs(R) - Pairs(S)$ and no other pairs. The merges all have zero cost, so $GMD_{f_m,f_s}(R, S) = |Pairs(R) - Pairs(S)|$.

Now we follow through with the derivation:

$$\begin{aligned}
1 - &\frac{GMD_{f_m,f_s}(R, S)}{GMD_{f_m,f_s}(R, \perp)} \\
&= 1 - \frac{|Pairs(R) - Pairs(S)|}{|Pairs(R)|} \\
&= 1 - \frac{|Pairs(R)| - |Pairs(R \cap S)|}{|Pairs(R)|} \\
&= \frac{|Pairs(R \cap S)|}{|Pairs(R)|} \\
&= PairPrecision(R, S) \qquad \square
\end{aligned}$$

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.

6 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 [36] 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 [37]. 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 base 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.

Surprisingly, the topic of efficiency of measure computation is not discussed in any ER paper. Fortunately in this paper, we propose an efficient algorithm that computes GMD in linear time for a large class of configurations. 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.

The algorithm we propose is called the Slice algorithm, which computes GMD when f_m and f_s are operation order independent functions. A complete description of the Slice algorithm is given in Section 7.2. The gist of the algorithm is to independently compute the cost of generating each cluster in S by splitting off the necessary components from clusters in R and then merging them together. For example, suppose $R = \{\langle a, c, e\rangle, \langle b, d, f\rangle\}$ and $S = \{\langle a, b, c\rangle, \langle d, e, f\rangle\}$. Cluster $\langle a, b, c\rangle$ must be generated by merging two fragments (one from each cluster in R): $\langle a, c\rangle$ and $\langle b\rangle$. The cost to split these fragments from their clusters in R is $f_s(2, 1) + f_s(1, 2)$, and the cost to merge them is $f_m(1, 2)$. When computing the cost to generate $\langle d, e, f\rangle$, we remember how many records have already been split from the clusters in R to properly compute the cost of splitting R further. In the example, no splits are necessary to get the fragments $\langle e\rangle$ and $\langle d, f\rangle$ because $\langle a, c\rangle$ and $\langle b\rangle$ have already been split off. So the cost to generate $\langle d, e, f\rangle$ is just $f_m(1, 2)$. We can then add the costs of generating these two clusters to obtain the total GMD.

We can compute the fragments needed to generate a cluster $\pi_S \in S$ by considering each record in π_S and looking up its location in R. We can then group the records by their R location to obtain the fragments. This step takes time $O(|\pi_S|)$ and thus the entire algorithm runs in time $O(N)$.

7 Computing Merge Distance

In this section we provide the details of algorithms for computing merge distance.

7.1 General Algorithm

A simple method for computing merge distance is the direct application of Dijkstra's algorithm. If we treat partitions as nodes in a graph and merge and split operations as the edges between nodes, then Dijkstra's algorithm will find a minimum cost path from R to S.

For this application, however, Dijkstra's algorithm would be highly inefficient. There are a few modifications to the algorithm that can help improve the performance. For one, Theorem 5.2 allows us to prune all paths that have a split after a merge has been performed. Further, at any state, we may consider if it is possible to reach the destination S through merges alone. If not, then more splits are required, so we need not consider any merges at this point. (See Lemma 5.2 for an explanation of this idea.)

Finally, we may make use of the monotonicity property of the merge and split cost functions to construct a lower-bound on the cost from any state to the destination S. Given this lower bound, we can apply the $A*$ algorithm instead of Dijkstra's algorithm, which may help narrow the search.

Unfortunately, we do not expect any of these optimizations to improve the exponential worst-case time for computing merge distance. We instead focus on a specific class of merge and split functions for which we have found an efficient algorithm.

7.2 Slice Algorithm

In this section, we describe a linear time algorithm called the Slice algorithm for computing generalized merge distance. 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

Algorithm 1 Slice algorithm

```
Input:   $R, S$: the result and gold standard. $R_i$ and
         $S_i$ are the $i$th clusters of $R$ and $S$ respectively
         $f_m, f_s$: the cost functions for 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 $cost$
```

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 $2 * f_s(2, 1) + f_m(2, 1)$.

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.

8 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 of the experiments demonstrates how the GMD measure can be configured using two important parameters: sensitivity to error type and sensitivity to cluster size. We then demonstrate the runtime performance of the Slice algorithm.

8.1 Data

To study the ER measures, we need an ER result R and gold standard G. These sets can either be synthetic or real. 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 will use both synthetic and real data. We first 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 base 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 base 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 R-Swoosh algorithm [8] uses a Boolean pairwise match function to compare records and a pairwise merge function to merge two records that match into a composite record. R-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 [9] 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.

8.2 Measure Conflicts

Most of the papers surveyed in Section 4 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 8.1. *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 $isBetter(M_2, R_2, R_1) = true$.*

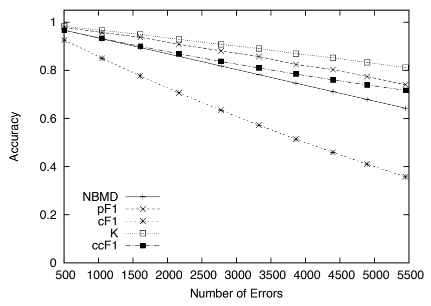
Measure conflicts occur because different measures evaluate different aspects of ER results. For example, pairwise precision only measures the portion of correctly matching base record pairs among the result while pairwise recall measures the portion of all correctly matching pairs found in the result.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

To see how frequently conflicts could occur, we first measure how "sensitive" the ER measures are for each error type in Figure 4. 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 4. As a result, the $NBMD$ value in one of the plots (in this case Figure 4(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.

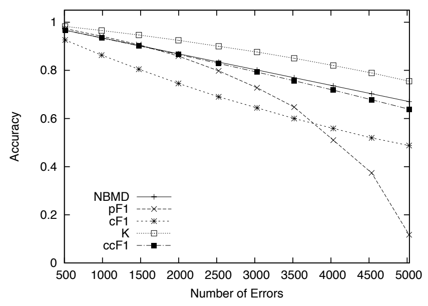
One can see from Figure 4 that if we confine ourselves to a single error type, any one measure is good enough to evaluate ER algorithms/results. That is, if R_2 has more errors than R_1, then $isBetter(M, R_1, R_2) = true$ for all measures M. In other words, there are no conflicts with unimodal errors.

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 4(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 4(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!

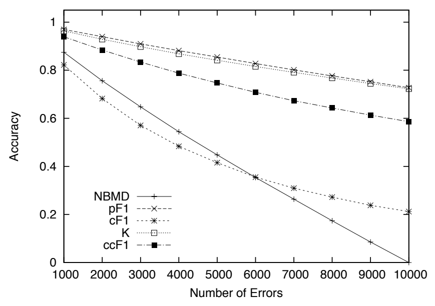
Figure 5 shows the number of conflicts that occur for each measure pair based on the data in Figure 4, sorted in decreasing numbers of conflicts. Each plot compares ER results of one error type to the ER results of another error type. Since there are 10 ER results for each error type according to Figure 4, we compare all 10×10 ER result pairs for each pair of ER measures. For example,



(a) Broken entity errors



(b) Glued entity errors



(c) Misplaced entity errors

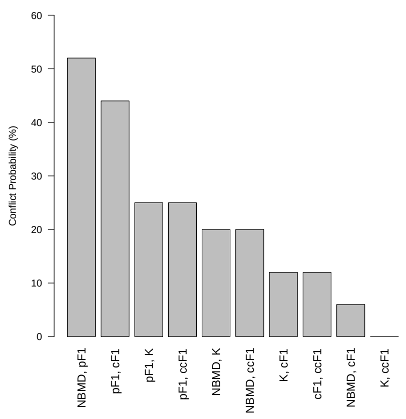
Figure 4: Sensitivity comparison for single error types

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 5). 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.

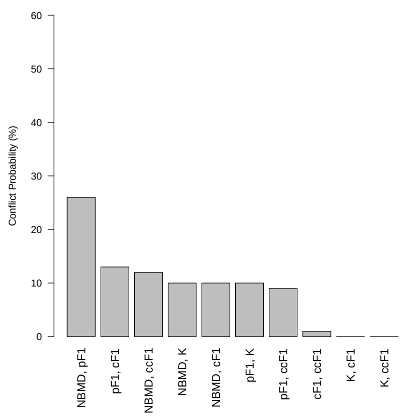
Conflicts can also occur in real datasets as shown in Table 4, which shows the measure results for the two ER algorithms run on the shopping and hotel datasets. We added the distance results for BMD and accuracy results for the other measures. It is important to understand that for BMD, a *smaller* distance indicates a more accurate ER result. For the hotel dataset, the Swoosh algorithm performs better than the Monge Elkan 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 ϵ = 0.01). For the shopping dataset, both pF_1 and BMD consider the Swoosh algorithm to be better than the Monge Elkan 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).

8.3 Configured GMD Results

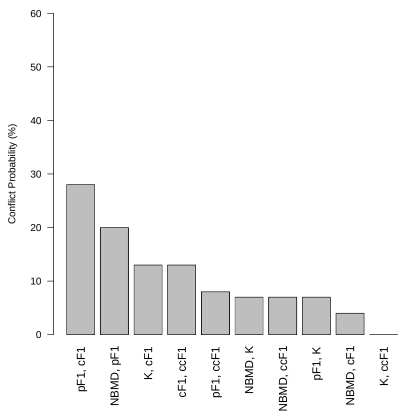
In this section, we show how GMD can be configured to measure accuracy in different ways. We first present the key parameters – sensitivity to error type and sensitivity to cluster size – that are configured in the GMD measure. We then experiment on various configurations of these pa-



(a) Glued and misplaced entity errors



(b) Broken and misplaced entity errors



(c) Broken and glued entity errors

Figure 5: Conflict frequencies

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
Monge Elkan	435	0.61	0.86	0.96	0.93
	Shopping				
Swoosh	29	0.86	0.98	0.98	0.99
Monge Elkan	34	0.73	0.98	0.97	0.99

rameters and show how each configured GMD measure can capture different qualities of ER results. Finally, we discuss how to choose the right measure for evaluating ER in a given application.

The GMD measure has two configuration "knobs": the merge and split cost functions f_m and f_s. The first parameter, sensitivity to error type, captures how sensitive the GMD measure is to broken entity errors and glued entity errors. If the GMD measure should be more sensitive to broken entity errors than glued entity errors, we can set f_m to return larger values than f_s, giving more penalty for records that should have been merged. Similarly, if the GMD measure should be more sensitive to glued entity errors, we can set f_s to return larger values than f_m. The second parameter, sensitivity to cluster size, captures how much penalty the GMD measure gives to larger clusters that need to be fixed. For example, we might want to make sure large clusters are properly resolved while tolerating errors in smaller clusters. To increase the sensitivity to cluster size, we can simply make both f_m and f_s return larger values.

Table 5 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 5.5). 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 5: 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 base records.

Figure 6 compares the sensitivities of the configured GMD measures in Table 5. We used the same gold standard and ER results used for Figure 4. Figures 6(a) and 6(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 6(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 6(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 6(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.

Choosing the Right Configuration With a configurable measure, a natural question is how to choose the right configuration for evaluating a given application. The selection can be done in two steps. First, determine the type of error that is most significant to the application. Second, determine whether resolving large clusters is more important than resolving smaller clusters. For example, one might be interested in correctly evaluating ER algorithms that emphasize good precision and also makes sure at least the large clusters are precisely resolved.

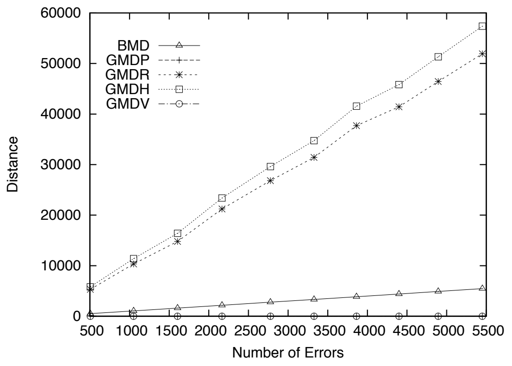
Table 6 shows how two ER algorithms can be compared using configured GMD measures. The ER results are identical to the ones used for Table 4. Each GMD measure gives certain information on how the two algorithms performed. For example, using the results of GMD_P and GMD_R, we know that the Swoosh algorithm is superior to the Monge Elkan 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 Monge Elkan, but similar BMD distances. (Recall that GMD_H is more sensitive to errors in large clusters than BMD.)

Table 6: Configured GMD results for real-world algorithms and datasets

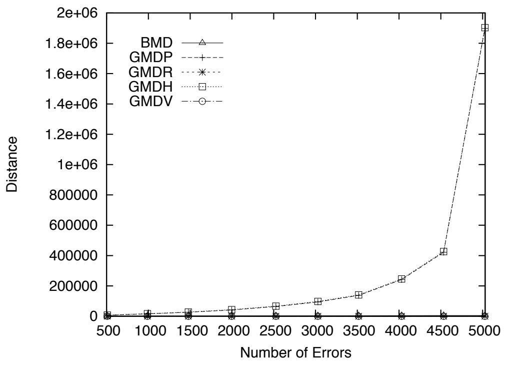
	BMD	GMD_P	GMD_R	GMD_H	GMD_V
	Hotel				
Swoosh	427	2087	158	2672	0.177
Monge Elkan	435	79	374	888	0.122
	Shopping				
Swoosh	29	28118	0	28147	0.084
Monge Elkan	34	0	12794	12828	0.078

8.4 Runtime Performance

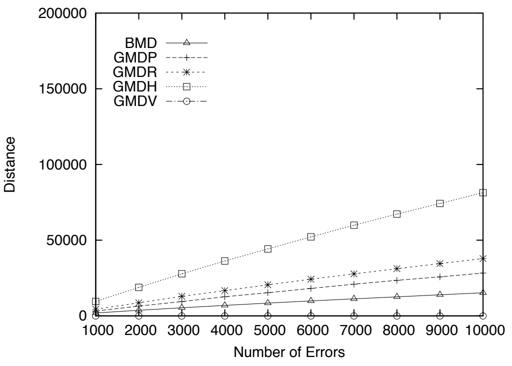
As discussed in Section 6, ER datasets can be huge, and the computation times for measures can be very signifi-



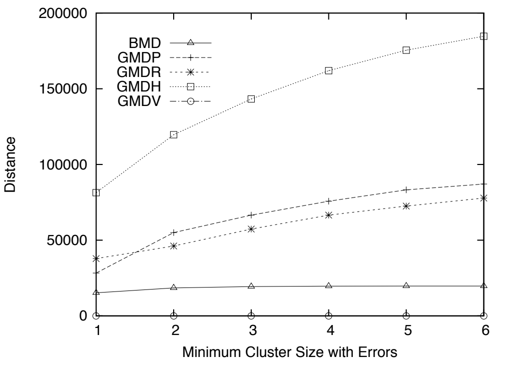
(a) Broken entity errors



(b) Glued entity errors



(c) Misplaced entity errors



(d) Misplaced entity errors in large clusters

Figure 6: Sensitivities of GMD measures for different error types

cant. 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 base 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 6) 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 7 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 base 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.

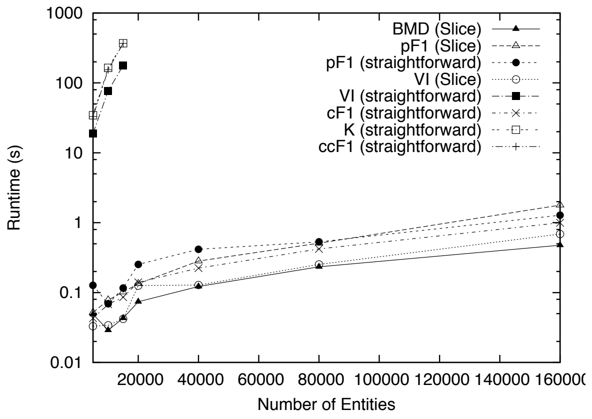


Figure 7: Scalability

9 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 also showed that GMD can be configured on two important parameters: sensitivity to error type and sensitivity to cluster size. As a result, one could more precisely define a measure that correctly evaluates 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.

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] L. Al-Hakim, *Information Quality Management: Theory and Applications*. Idea Group Inc.

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

[6] E. Ukkonen, "On approximate string matching," in *FCT*, 1983, pp. 487–495.

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

[8] 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.

[9] A. E. Monge and C. Elkan, "An efficient domain-independent algorithm for detecting approximately

duplicate database records," in *Proc. of SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997, pp. 23–29.

[10] Wikipedia, "Metric (mathematics) — Wikipedia, the free encyclopedia," 2009, [Online; accessed 11-June-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Metric_(mathematics)&oldid=295576078

[11] 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.

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

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

[14] S. Tejada, C. A. Knoblock, and S. Minton, "Learning domain-independent string transformation weights for high accuracy object identification," in *KDD*, 2002.

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

[16] W. E. Winkler and Y. Thibaudeau, "An application of the fellegi-sunter model of record linkage to the 1990 u.s. decennial census," in *U.S. Decennial Census. Technical report, US Bureau of the Census*, 1987.

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

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

[19] 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.

[20] 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.

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

[22] W. W. Cohen and J. Richman, "Learning to match and cluster large high-dimensional data sets for data integration," in *KDD*, 2002, pp. 475–480.

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

[24] M. Bilenko, R. J. Mooney, W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "Adaptive name matching in information integration." *IEEE Intelligent Systems*, vol. 18, no. 5, pp. 16–23, 2003.

[25] A. Culotta and A. McCallum, "Joint deduplication of multiple record types in relational data," in *CIKM*, 2005, pp. 257–258.

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

[27] S. Sarawagi and A. Bhamidipaty, "Interactive deduplication using active learning," in *KDD*, 2002.

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

[29] Y. Song, J. Huang, I. G. Councill, J. Li, and C. L. Giles, "Efficient topic-based unsupervised name disambiguation," in *JCDL*, 2007, pp. 342–351.

[30] D. Ramage, P. Heymann, C. D. Manning, and H. Garcia-Molina, "Clustering the tagged web," in *WSDM*, 2009, pp. 54–63.

[31] H. Pasula, B. Marthi, B. Milch, S. J. Russell, and I. Shpitser, "Identity uncertainty and citation matching," in *NIPS*, 2002, pp. 1401–1408.

[32] 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.

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

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

[35] 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.

[36] 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.

[37] 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.