Refining Duplicate Detection for Improved Data Quality

Yu Huang and Fei Chiang

Dept. of Computing and Software
McMaster University
{huang223, fchiang}@mcmaster.ca

Abstract. Detecting duplicates is a pervasive data quality challenge that hinders organizations from extracting value from their data sooner. The increased complexity and heterogeneity of modern datasets has lead to the presence of varying record formats, missing values, and evolving data semantics. As data is integrated, duplicates inevitably occur in the integrated instance. One of the challenges in deduplication is determining whether two values are sufficiently close to be considered equal. Existing similarity functions often rely on counting the number of required edits to transform one value to the other. This is insufficient in attribute domains, such as time, where small syntactic differences do not always translate to 'closeness'. In this paper, we propose a duplication detection framework, which adapts *metric functional dependencies (MFDs)* to improve the detection accuracy by relaxing the matching condition on numeric values to allow a permitted tolerance. We evaluate our techniques against two existing approaches using three real data collections, and show that we achieve an average 25% and 34% improvement in precision and recall, respectively, over non-MFD versions.

1 Introduction

Data quality has become a pervasive challenge for many industries and organizations, where the time to reap value from the data has been hindered by missing values, inconsistencies, and duplicate records. Data duplication occurs when a real-world entity has two or more different representations within or across databases. Ideally, in an error-free system with perfectly clean data, each record in a database has a unique identifier. Unfortunately, in most practical cases, this does not occur, and the data often lacks a unique, global identifier. This especially occurs in data integration when data is integrated from multiple sources across different departments or organizations. In such cases, it is inevitable to introduce duplicates due to differences in record format, standardizations, schema and typos. Hence, identifying duplicates in the data is a critical step towards ensuring reliable and trusted data.

However, identifying duplicates is not a trivial problem. Varying record semantics, syntax, even the frequency of terms may affect the accuracy of identified duplicates. Existing solutions use string similarity to determine whether

two records are duplicates [11]. This process is far from perfect. String similarity metrics consider all values as strings, and typically measure the number of edits needed to transform one string to the other. This is not suitable for numeric values, such as time, where small character differences can lead to large differences in semantics. For example, the difference between times `'2:02pm'` and `'12:02pm'` is only one character, but the actual semantic difference is 2 hours. By using Euclidean distance, we can capture the actual difference between two numeric values, but if users have data quality requirements that state a relationship between attributes in a database (e.g., movie theater and movie title determine the movie start time), we need a mechanism that allows us to specify data quality rules. These data quality rules can be used as additional information to improve the accuracy of the duplicate detection task.

ID	SRC	FLIGHT	DEP_CTY	DEST_CTY	DUR	DEP_TM	ARR_TM	TERM	GATE
r_1	flightexplorer	AA-1007-TP-MIA	Tampa	Miami	62	13:55	14:57	T1	D5
r_2	airtravelcenter	AA-1017-TPA-MIA	Tampaa	Miami	56	2:07 PM	3:03PM	T1	D5
r_3	myrateplan	AA-1007-TPA-MIC	Tampi	Miami	52	14:03	14:55	T1	D5
r_4	helloflight	AAA-1007-TPA-MIA	Tampe	Miama	54	14:03	14:57	T1	D5
r_5	flightexplorer	AA-1518-YVV-SDF	Vancouver	Louisville	417	12:08	19:02	T2	A15
r_6	airtravelcenter	AA-1588-YVR-SDF	Van	Louis	418	12:09 PM	7:00 PM	T2	A15
r_7	myrateplan	AA-1518-YVR-SDF	Vancouver	Louisville	432	12:13	19:25	T2	A15
r_8	helloflight	AAA-1518-YVR-SDF	Van	Louisvilla	430	12:10	19:22	T2	A15

Table 1: Sample flights data [1]

Example 1. Consider Table 1 containing a sample of real flight records for December 1st, 2011, collected from various online sources that state the departure (DEP_TM) and arrival (ARR_TM) times of flights (FLIGHT) from a departure city (DEP_CTY) to a destination city (DEST_CTY), with an expected flight time duration (DUR) in minutes [1]. Upon observation, we can loosely recognize that these records are likely duplicated based on two identifying flights. The data contains several typos in FLIGHT, different representations in DEP_CTY and DEST_CTY using short forms, and airport codes. In addition, the attributes DUR, DEP_TM, and ARR_TM all have different values. The challenge is to automatically determine which records are actually duplicates.

Existing approaches compare these attributes using string similarity, and Euclidean distance, and check whether the values are sufficiently close to be identified as duplicates. These methods determine their similarity (or distance) based on the syntactic forms of the values, and ignore any semantic relationships among attributes. The drawback of this approach is that checking if two values are 'close enough' is often arbitrary, and subjective depending on the analyst. In addition, correlations and dependencies among attributes often play an important role in determining whether the records are clean.

In our example, suppose we have a data quality rule that states the DEP_CTY and DEST_CTY determine the DUR. Intuitively, a departure and destination city pair will give a unique flight duration. Existing database dependencies, such as functional dependences are not robust enough to capture these functional

relationships from heterogeneous sources as they require the duration times to be exactly equal. In Table 1, not all the flight durations between `Tampa` and `Miami` are equal. While these may be in error, it is also possible that different sources have different interpretations of departure time (and arrival time) leading to different duration times. For example, some sources may consider departure time as the time the aircraft leaves the gate, while others will consider it the time the aircraft is off the ground. Due to these variations across heterogeneous sources, the data contains small discrepancies that are tolerable.

In this paper, we propose a deduplication framework that uses a class of data quality rules, called *metric functional dependencies* (MFD) that allows for small variances in numeric values during the matching process [15]. We show that by using these data quality rules, deduplication accuracy can be improved. We make the following contributions:

1. We design a deduplication framework that uses *metric functional dependencies* [15] to specify relationships among attribute values, and allows small metric differences in numeric values. This helps to improve overall accuracy when identifying duplicates.
2. During the matching process, we propose a weighting scheme that differentiates terms (in a record) based on their frequency of occurrence.
3. We evaluate our approach against two existing solutions using real data collections. We show that our solution achieves improved accuracy.

2 Preliminaries

2.1 Functional Dependencies

Our model assumes that the data is in a relational (tabular) format. Metric functional dependencies (MFDs) are based on the same intuition as functional dependencies (FDs). For a relation R, a functional dependency (FD) $F : X \rightarrow Y$ is a constraint between two attribute sets X and Y. A data instance I of R satisfies F, denote $I \vDash F$, if for every pair of records r_1, $r_2 \in I$, if $r_1[X] = r_2[X]$ then $r_1[Y] = r_2[Y]$. Records that do not satisfy F are said to be *inconsistent* w.r.t. F, or that they *violate* F.

Example 2. In Table 1, the FD F_1: [GATE] $\rightarrow$ [TERM], states that a given GATE determines which TERMINAL it belongs to. F_1 is satisfied, since for every unique value in attribute GATE, we observe a unique value in TERM.

2.2 Metric Functional Dependencies

For a relation R, a *metric functional dependency (MFD)* defines a relationship between X and Y, where X and Y are attribute sets in R [15]. In general, we can say that the MFD $X \xrightarrow{\delta} Y$ with metric d holds if for any two records r, r', if $r[X] = r'[X]$, then we have $max(d(r[Y], r'[Y])) \leq \delta$, where δ is a tolerance

parameter. Compared to FDs, MFDs relax the equality condition on the consequent (Y) attributes by allowing small differences. We can define the "closeness" of these values by a metric function d.

Example 3. In Table 1, records r_5 and r_7 do not satisfy FD F_2 : [DEP_CTY, DEST_CTY] $\rightarrow$ DUR since r_5[DUR] = 417 and r_7[DUR] = 432. However, if we allow a variance of $\delta = 15$, then r_5, r_7 are no longer violations, and we can define MFD σ : [DEP_CTY, DEST_CTY] $\xrightarrow{\delta=15}$ [DUR] as a data quality rule on Table 1.

3 Framework Overview

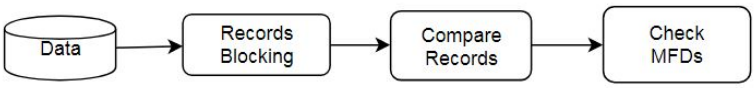

Fig. 1: Framework modules.

Figure 1 presents an overview of our framework. In order to identify duplicates, we need to compare records pairwise to calculate their similarity score. For a database D, in the worst case, we potentially need to compare records pairwise which involves $|D|^2$ comparisons. This approach is computationally infeasible for large datasets. To avoid this, we partition the records into disjoint blocks and compare the records pairwise within a block. This reduces the overall number of comparisons significantly. Assume D can be partitioned into m blocks, then the number of comparison is $|D|(|D|/m - 1)$. In this paper we use the k-means unsupervised learning algorithm to cluster the records into blocks [19].

To cluster records, we consider each record as a document, and calculate the tf-idf score for each term that appears in a record r. After this, each record $r \in D$ can be converted to a tf-idf vector $\boldsymbol{v}$, which can be considered as a n-dimensional vector. The k-means clustering can partition these vectors into k sets to minimize the variance among them. We compute term frequencies to automatically differentiate terms that will help to improve the accuracy in identifying duplicates.

After record blocking, we need to compare records within a block. We take a bottom-up approach that calculates the similarity of terms within a record, and aggregate these scores to compute a similarity score between two records. During term comparison, we distinguish terms based on their frequency of occurrence in each block. We assume that terms occurring frequently contributes less towards the overall score, and are not as good indicators of record similarity.

Finally, given a set of candidate duplicates, we apply an MFD σ_δ to refine the set of matched duplicates by allowing small tolerances in the consequent attribute values to provide greater flexibility during the matching step.

4 Duplicate Detection with MFDs

In this section, we present details of our framework, including the matching functions we use, followed by the weighting scheme that differentiates terms during the record comparison step. We then discuss how MFDs are used to identify a greater number of duplicates.

4.1 Matching Functions

To deduplicate two records r and r', we need to compare relevant terms from each record, and aggregate the similarity scores to determine whether they are sufficiently similar as duplicates. We assume a mapping exists to compare term $t_i \in r$ with $t'_i \in r'$. Hence, we can use similarity metrics to compute the similarity between t_i and t'_i. We use two similarity functions in our framework: edit distance and q-grams based matching. We note that while we select these two functions, our framework supports other matching functions, and is agnostic to the type of similarity function used.

The edit distance is a common metric to measure the distance between terms, and it denotes the minimum number of edit operations to transform one term to another. Let $\text{ed}(t_i, t'_i)$ represent the similarity between terms t_i, t'_i using edit distance, defined as $sim_{ed}(t_i, t'_i) = 1 - \frac{editDist(t_i, t'_i)}{max(|t_i|, |t'_i|)}$.

The use of q-grams is a commonly used approach that converts a term to a set of q-grams, and compares terms via comparing their q-gram sets. The intuition is that if two strings are similar, they share a large number of q-grams. To generate a q-grams set, we consider a sliding window of size q over r, and use the sequence of characters within the window as a token. In this paper, we use $q = 2$, and generate bi-gram tokens. For example, to compare `'paper'`, `'pepper'` and `'page'`, we divide each term into bi-gram tokens, and use a bi-gram bitmap to determine whether a bi-gram appears in a value or record as showed in Table 2.

It is efficient to calculate the similarity score between two values via their bitmaps. Let b_t and $b_{t'}$ represent the bitmap of record r and r' respectively, then the Jaccard similarity[13] between two records can be defined as $Jacc(r, r') = \frac{b_r \cap b_{r'}}{b_r \cup b_{r'}}$, where $\cap$ denotes bitwise AND operation, and $\cup$ denotes the bitwise OR. operation. Consider two terms $t =$ `'paper'` and $t' =$ `'pepper`, the corresponding bitmaps are $b_t = [1, 1, 1, 1, 0, 0, 0, 0]$ and $b_{t'} = [0, 0, 1, 1, 1, 1, 0, 0]$, and their Jaccard similarity is $Jacc(t, t') = 2/6$.

	pa	ap	pe	er	ep	pp	ag	ge
paper	1	1	1	1	0	0	0	0
pepper	0	0	1	1	1	1	0	0
page	1	0	0	0	0	0	1	1

Table 2: Bigrams bitmap.

4.2 Differentiating Record Terms

Deduplicating two records r and r' involves comparing relevant terms from each record, and aggregating the similarity scores (such as those described in the previous section) to determine whether the terms are sufficiently similar. Existing

techniques have primarily assumed that the terms appearing in a record r have equal weight during the aggregation. For example, in a stock dataset containing company names, terms such as `'Apple'` occur less frequently in the data than terms such as `'Corporation'` and `'Ltd.'`. In our framework, we distinguish these less frequent terms during the aggregation process by placing more emphasis terms such as `'Apple` since they serve as a more accurate indicator of similarity than commonly occurring terms in the data. We define a weight w_{ij} for each pair of compared terms t_i and t_j, using the average term frequency, as $w_{ij} = \log \frac{freq(t_i)+freq(t_j)}{2}$.

Combining the weights with the term similarity scores, we compute a similarity score between two records r and r' as:

$$sim(r, r') = \sum_{i,j \in n} w_{ij} \cdot c(t_i, t_j) \tag{1}$$

where $c(t_i, t_j)$ is a function that returns the similarity score between terms t_i and t_j, such as edit-distance or q-grams similarity, discussed earlier. Intuitively, this weighting scheme identifies core terms in a record, and places greater importance on these core terms during the similarity matching, while discounting terms occurring with high frequency.

4.3 Matching with MFDs

During the record matching, we compare two terms t_i and t_j, and compute a similarity score $c(t_i, t_j)$. When the terms are numeric and small variances are allowed, as specified via an MFD, We refine $c(t_i, t_j)$ based on the δ value. Specifically, if $\max(d(t_i, tj)) \leq \delta$, then we consider the values t_i, t_j to be equal, as per the MFD. This allows us to include a broader set of records that qualify as duplicates based on allowing small differences in Y for an MFD $\sigma : \mathrm{X} \xrightarrow{\delta} Y$. Setting δ is application dependent as each domain defines their allowable variances. In future work, we intend to evaluate the performance impact of varying δ values.

Example 4. Continuing our example from Table 1, if $r_1 - r_4$ are considered sufficiently similar in attributes `DEP_CTY, DEST_CTY`, then we can apply the given MFD σ : [DEP_CTY, DEST_CTY] $\xrightarrow{\delta=15}$ [DUR]. Under existing string or Euclidean comparison functions, these records would be considered distinct given the range of `DUR` values. However, by applying σ, we validate that all flight time durations are within 15 minutes. We update the similarity scores for $r_1 - r_4$ to equal 1, which strengthens the record similarity scores between these tuples.

Furthermore, by having knowledge of attribute dependencies via σ, our framework supports differentiation among attributes to consider attribute sets XY (participating in σ) more strongly during the matching process. As the example above shows, by leveraging MFDs, we expand the scope of potential duplicates to include those containing numeric values with small variances that otherwise would not have been captured.

5 Evaluation

We conduct preliminary experiments to investigate the accuracy of our techniques. We focus on: (1) the benefits of using MFDs during the matching process; and (2) the comparative accuracy of our term differentiation and MFD matching against two existing solutions, WHIRL [10], and SERF [8].

5.1 Datasets

We use three real datasets in our evaluation: (1) flights data containing 27000 records showing flight arrival and departure times for various airlines collected from 38 online sources [1]; (2) stock listings showing company names, IPO dates from the NASDAQ exchange with 8744 records and 8 attributes [3]; (3) restaurants data containing 864 records and 5 columns listing restaurant names, addresses, and category [2]. For our performance evaluation, we use the UIS database generator to control the scalability in the number of tuples [4].

We compare the identified duplicates against the ground truth, and compute precision and recall defined as, $precision = \frac{\#\text{correct duplicates returned}}{\#\text{total duplicates returned}}$ and $recall = \frac{\#\text{correct duplicates returned}}{\#\text{true duplicates}}$.

5.2 Benefits of Matching with MFDs

We use the flights dataset to experimentally verify the benefits of including MFDs during the record matching step. We compute the precision and recall of our approach using edit distance, and the q-grams based functions, referred to as *Weighted Frequency Edit Distance (WFED)*, and *Weighted Frequency Bigrams (WFB)*, respectively. Figure 2 and Figure 3 show the precision and recall results, respectively, of our solution with and without applying MFDs. We observe that applying MFDs improves both precision and recall. For precision, we achieve a 24.6% (WFED), and 25.2% (WFB), improvement over the non-MFD versions, showing that about a quarter of the records contained numeric values where non-equality comparisons were needed. This is especially evident in the flights dataset which contained several numeric attributes measuring flight time durations, departure, and arrival times.

For recall, we achieve even larger gains of 33% (WFED), and 36% (WFB) over the non-MFD versions due to the increased number of records we are able to capture by relaxing the strict equality conditions. In summary, our initial results are promising, showing significant gains in accuracy by allowing terms to be matched within a small variance. As next steps, we are exploring how the concept of MFDs can be applied to categorical and string data (currently MFDs only apply to numeric data). This will enable us to capture duplicate records involving non-numeric attributes, and improve our accuracy rates.

5.3 Sensitivity to Similarity Levels

We evaluate the sensitivity of our techniques as we vary the similarity threshold ϑ. Figure 4 and 5 show the precision and recall, respectively. As expected, we observe that all techniques achieve greater precision as the similarity increases due to more stringent matching criteria to find true duplicates. In contrast, recall values decline for increasing similarity values due to the stringent pruning that may miss some duplicates. Our term differentiation allows WFED to achieve improved accuracy. For example, in previous solutions, compared strings such as `'IBM Corp'` and `'ABM Corp'` would consider compared terms `('IBM', 'ABM')`, and `('Corp')` equally during the similarity matching. Since we weigh `('IBM', 'ABM')` more heavily, greater emphasis is placed on these terms to determine whether the records are duplicates, helping to increase overall accuracy.

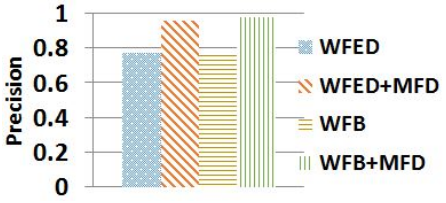

Fig. 2: Comparative precision.

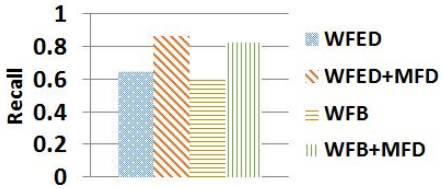

Fig. 3: Comparative recall.

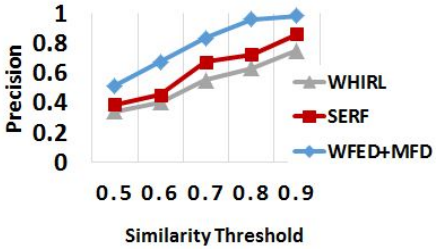

Fig. 4: Precision for varying ϑ

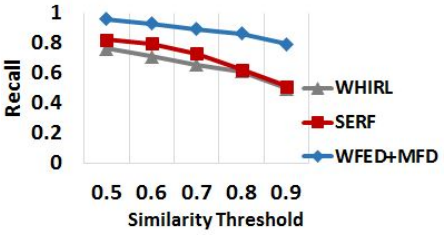

Fig. 5: Recall for varying ϑ

6 Related Work

Record deduplication, also referred to as *record linkage* or *entity resolution* contains an extensive literature of existing techniques [14, 9, 5]. We first discuss existing similarity metrics used in deduplication, followed by a brief discussion of approximate string matching, and then discuss the WHIRL and SERF techniques that we consider in our experiments.

Similarity Metrics. Selecting a suitable similarity metric according to application needs, and the relevant data types, is a key step towards achieving good accuracy. In addition to edit distance, and q-gram based methods, the Smith-Waterman distance is commonly used, and computes the longest common

substring that exists between two strings [17]. The affine gap distance metric introduces two additional edit operations (*open gap, extend gap*) that extend the transformation operations in edit distance [20]. The Jaccard similarity metric compares two sets through the intersection of shared elements [13], while cosine similarity computes the cosine of the angle between two record vectors [18].

Approximate String Matching. Approximate string matching techniques have used cluster-based approaches that take advantage of co-citation or co-occurrence information of values to identify plausible duplicates [6, 7]. Hassanzadeh et. al., propose a evaluation framework named Stringer that evaluated a series of clustering algorithms [12]. Menestrina et al. proposed a *generalized merge distance* which uses cluster split and merge operations to measure the distance between two records [16].

WHIRL. Cohen presents a system named WHIRL which uses tf-idf weights with cosine similarity to calculate string similarity [10]. WHIRL divides a string s into a set of terms W, and for each term $w \in W$, the term frequency (tf) to inverse document frequency (idf) score is used to represent w. The tf-idf value is computed as $tf \cdot idf = \log(tf + 1) * \log(idf)$. tf is the number of times that w appears in s, and idf is $|D|/N$, where N is the number of records in the database D that contain w, and $|D|$ is the total number of records in D. After each term w is represented as a tf-idf component, s can be represented as a vector, and the similarity score between two records can be calculated by applying cosine similarity to two vectors. Their method mainly relies on the distribution of terms to measure records similarity, but does not look into each term s to check the characters which will lead to ignore spelling errors. While our framework also considers the distribution information of attribute values, we also consider using the term frequencies to distinguish the importance of each term.

SERF. The Stanford Entity Resolution Framework (SERF) identifies similar records via match and merge functions [8]. They develop a generic infrastructure for Entity Resolution (ER). The goal of ER is to "resolve" entities, by identifying the records that represent the same entity and reconciling them to obtain one record per entity. Their system defines a *match* function that takes two records as input and returns true if the input records represent the same entity. The *merge* function aggregates the duplicate records into a consolidated record representing the entity. The authors propose the SWOOSH algorithm to reduce the number of match and merge operations. This work shares the same spirit as ours, while we consider finer grained matching for each term to distinguish column values during the matching process. Specifically, we do not assume all values within a record are treated equally, and our framework automatically computes the weights to differentiate terms in a record.

7 Conclusion and Future Work

Duplicate records remain a pervasive data quality challenge that is fueled by the increasing complexity, size, and heterogeneity of modern datasets. Real data often contains missing values, inconsistent data formats, and unclear seman-

tics. We present a deduplication framework that leverages term frequencies as a differentiator, and exploits *metric functional dependencies (MFDs)*, during the similarity matching process to improve duplicate detection. Our evaluation shows that our approach achieves improved accuracy over two existing methods, and significant gains in accuracy when MFDs are used during the matching. In future work, we intend to improve the robustness of our framework against missing values by developing imputation methods that impute the missing values. We also intend to expand our model to include semi-structured data, provide guidelines for determining a suitable value for the tolerance parameter δ, and conduct more extensive scalability experiments using large data collections.

References

1. flight dataset. http://lunadong.com/fusionDataSets.html.
2. restaurant dataset. https://www.cs.utexas.edu/users/ml/riddle/data.html.
3. stock dataset. http://www.nasdaq.com/. [Online; accessed 20-November-2016].
4. UIS Database Generator. https://www.cs.utexas.edu/users/ml/riddle/data.html.
5. Overview of record linkage and current research directions. http://www.census.gov/srd/papers/pdf/rrs2006-02.pdf, 2006.
6. N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.
7. J. A. Aslam, E. Pelekhov, and D. Rus. The star clustering algorithm for static and dynamic information organization. *J. of Graph Algs. and Apps.*, 8:95–129, 2004.
8. O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. Whang, and J. Widom. Swoosh:generic approach to entity resolution. *VLDBJ*, 18(1):255–276, 2009.
9. I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5, 2007.
10. W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD Record*, volume 27, pages 201–212, 1998.
11. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1), 2007.
12. O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.
13. P. Jaccard. The distribution of the flora in the alpine zone. *New phytologist*, 11(2):37–50, 1912.
14. W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, 24(12):12–18, 1991.
15. N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *ICDE 2009*, pages 1275–1278, 2009.
16. D. Menestrina, S. E. Whang, and H. Garcia-Molina. Evaluating entity resolution results. *Proceedings of the VLDB Endowment*, 3(1-2):208–219, 2010.
17. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.
18. M. Steinbach, G. Karypis, V. Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526, 2000.
19. K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, et al. Constrained k-means clustering with background knowledge. In *ICML 2001*, volume 1, pages 577–584, 2001.
20. M. S. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(3):367–387, 1976.