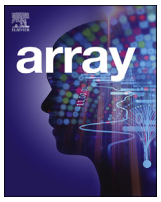
Array 11 (2021) 100070

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/journals/array/2590-0056/open-access-journal



All-Three: Near-optimal and domain-independent algorithms for near-duplicate detection

Aziz Fellah

School of Computer Science and Information Systems, Northwest Missouri State University, Maryville, MO, 64468, USA

ARTICLE INFO

Keywords:
Near-duplicate detection
Near-duplicates
Approximate duplicates
Clustering
Data mining applications and discovery
Data cleaning

ABSTRACT

In this paper, we propose a general domain-independent approach called Merge-Filter Representative-based Clustering (*Merge−Filter−RC*) for detecting near-duplicate records within a single and across multiple data sources. Subsequently, we develop three near-optimal classes of algorithms: constant threshold (*CT*) variable threshold (*VT*) and function threshold (*FT*), which we collectively call *All−Three* algorithms. Merge−Filter−RC and All−Three mold the basis of this work. Merge−Filter−RC works recursively in the spirit of divide-merge fashion for distilling locally and globally near-duplicates as hierarchical clusters along with their prototype representatives. Each cluster is characterized by one or more representatives which are in turn refined dynamically. Representatives are used for further similarity comparisons to reduce the number of pairwise comparisons and consequently the search space. In addition, we segregate the results of the comparisons by labels which we refer to as very similar, similar, or not similar. We complement All−Three algorithms by a more thorough reexamination of the original well-tuned features of the seminal work of Monge-Elkan's (*ME*) algorithm which we circumvented by an affine variant of the Smith-Waterman's (*SW*) similarity measure.

Using both real-world benchmarks and synthetically generated data sets, we performed several experiments and extensive analysis to show that All−Three algorithms which are rooted in the Merge−Filter−RC approach significantly outperform Monge-Elkan's algorithm in terms of accuracy in detecting near-duplicates. In addition, All−Three algorithms are as efficient in terms of computations as Monge-Elkan's algorithm.

1. Introduction

The problem of identifying whether multiple representations of a real-world entity or object are the same has been originally defined by NewCombe et al. [1]. Since then, this long-standing problem has been studied extensively in computer science and related fields under various names using a multiplicity of terminology; just to name, record linkage [2,3] in the statistics community, approximate matching [4] in information retrieval, entity resolution [5], object identification [6] in machine learning, merge/purge [4,7,8] and near-duplicate detection [9–18] in databases and algorithms. In large and various databases, a major task in a data cleaning process is identifying sets of records that are semantically duplicates of each other, but not syntactically identical. Such type of duplicate records are also referred to as similar, approximate or near-duplicates in research literature. In the context of this paper, we adopt the last terminology, that is, near-duplicates. The most common variations in representing the same entity (*i.e.,* records, objects) with a multitude of representations can primarily arise from typographical errors, misspellings, missing data, and differences in abbreviations and schemes, as well as the integration of multiples data sources into a single data set. In general in data cleaning, near-duplicates may exist within a single source, whereas in data integration near-duplicates may exist within or across various data sources. However, both cases have the same common goal, detecting near-duplicates and the closeness of similarity among entities in a large collection accurately and efficiently. That is, to determine which records/objects in the same or different databases refer to the same underlying real-world entity.

For example, consider the following references that have been recorded at three different colleges, *"Jeff David Ullman Stanford University, Dept. of Computer Science"*; *"Jeffrey D. Ullman Computer Science Dept., Stanford Univ., CA, USA"*; *"J. D. Ullman, Department of Computer Science, Stanford University, USA"*. All the three references refer to the same individual, even though they are quite different if byte-by-byte comparisons are used. It is often the case that data in different repositories hold information regarding identical entities, but might be stored in different formats and schemes which may result in a possible data inconsistency and nonconformity. One example would be identifying authors and citations in a bibliography database such as DBLP, CompuScience, and

E-mail address: afellah@nwmissouri.edu.

https://doi.org/10.1016/j.array.2021.100070
Received 20 February 2021; Received in revised form 5 May 2021; Accepted 12 May 2021
Available online 27 May 2021

CiteSeer. Several algorithms, in particular domain-dependent, for quantifying the degree of similarities have received particular attention in a wide range of applications, including web search engines and mining, medical and census data, plagiarism and spams, mailing list deduplication, and image database. A substantial body of research has been conducted in a spectrum of domains, see for example [19–28].

2. Background and related work

Naive near-duplicate detection methods which are based on comparing every pair of records become intractable in the context of huge data collections such as social networks. For example, Facebook with over 2.97 billion active users worldwide, results in more than 2.45 $\times$ 10^{19} comparisons. Several methodologies and solutions have been proposed in research to reduce the total computational cost and improve the accuracy. In a stand-alone literature review, a variety of blocking methods which are based on the selection of a key (*blocking key*) for generating subsets of potential near duplicates have been investigated in research [5,11,29]. The main focus of these methods is that records having the same key value should be added to the same block and labeled as potential near-duplicates for further analysis. Jaro and Winkler [3,24,30] extended the key-based approach by using expressions with multiple keys which increases the accuracy and reduces the number of false matches and false misses. Such keys can be adjusted interactively by trial-and-error analysis to achieve the best results. However, the process of finding a perfect key is often difficult and requires good domain knowledge. Other related blocking criteria based on applications' characteristics (*i.e.,* medical, census, bibliography data) are also used as a first-step preprocessing to identify initial blocking. In general, blocking algorithms are of low time complexity, but with several drawbacks. For example, records with minor typographical errors or simple misspelling are clustered in different blocks. It has been observed in Refs. [23,24] that 40–70% of the matches are found in the first blocking pass. By the fourth blocking pass, 0.0001% of the pairs are actually duplicates. These results show that the proportions of pairs that were duplicates in successive blocking criteria fell at an exponential rate. A well-known duplicate detection framework known as sorted neighborhood method (*SNM*) was proposed by Hernandez and Stolfo [8,31] and is based on that near-duplicates tend to localize in the neighborhood. The algorithm uses blocking to sort all records based on a sorting key and then slides sequentially a window of fixed size (*sliding window*) over the sorted records. The window uses the blocking scheme by including all records with similar keys (lexically nearby keys) in the same window. All records within each sliding window are considered as potential near-duplicates and then compared with each other, however, the window size is difficult to set. Overall, the sliding window is a more robust approach than other techniques in improving the near-duplicate detection accuracy, but it is likely to fail in grouping similar records outside the window size if substantial typographical errors occurs in the first characters of the sorting key. Moreover, there is substantial research that has been proposed as an umbrella for *SNM* where blocking has been empirically evaluated on different domains and data sets. Some of these extended variations of *SNM* and blocking approaches outperform the basic *SNM* and blocking approaches in terms of reducing the complexity cost and improving the accuracy. For instance, Yan et al. [7] proposed an adaptive variant scheme of *SNM* for record linkage by adjusting the size of the sliding window dynamically during the execution time to build non-overlapping blocks, but it has been confirmed that their work did not outperform the original *SNM*. Other complicated near-duplicate identification techniques such as *q*-gram, iterative blocking, overlapping blocking, multiple blocking, token-based, learning process, and domain knowledge, with some assumptions on the entities have been investigated in the literature, just to name few [5,25,32–34].

Furthermore, with rapid advances in big data computing and web era, near-duplicate detection research is expanding in many ways to industry applications such as managing massive documents, and extracting insights and multi-dimensional information from business, financial (*i.e.,* credit cards), and healthcare data. A comprehensive evaluation and an umbrella of techniques have been investigated. A common factor between these algorithms is they cannot guarantee finding all near-duplicates and also cannot guarantee the accuracy. Vogal et al. [35] provided an annealing standard to evaluate near-duplicate detection results. In other words, the accuracy and completeness of duplicates should converge incrementally and interatively to a gold or silver standard that defines which records represent the same real-world entities.

Near-duplicate detection algorithms mainly focus on effectiveness and efficiency, but not on scalability which has been addressed by Naumann et al. [36], who assign appropriate similarity measures to attributes based on their semantics. For instance, names of persons should be compared differently than email addresses even though they belong to the same string data type. Overall, most of these methods require a good understanding of the application domain and a supervised user interaction for refining and adjusting parameters such as distance functions, window size, and thresholds.

The most predominant domain-independent algorithm for near-duplicate detection is that of Monge-Elkan (*ME*) [4,14]. This seminal work is based on stretching adequately the *SNM*'s sliding window [8] that holds a fixed number of record sets and grouping records into clusters.

Monge-Elkan's algorithm has a much more improved efficiency over many duplication methods including the *SNM* algorithm, but has the same detection accuracy as the *SNM* and performs even far better than a large number of other algorithms in terms of time and accuracy. In addition, a spectrum of similarity and distance measures, highly dependent on the application domain, have been investigated in the literature and are certainly not a new area of research. They range from fairly simple schemes to more complex well-tuned edit distances. The most prominent class of character-based metrics known as edit distances are Levenshtien, Jaro-Winkler, and Smith-Waterman similarity measures [3,30,37]. One extension to the Levenshtein distance is the Needleman-Wunsch [38], which additionally allows variable substitution's cost for different characters. That is, it provides a mapping for each pair of symbols (*i.e.,* characters) from the alphabet to some cost. Other effective similarity measures, token-based and hybrid metrics, have been also investigated in the literature, for example, Jaccard, *n*-grams, Cosine, Monge-Elkan, and natural language processing techniques (*i.e.*, TF-IDF) similarity measures. Similarity metrics range from fairly simple schemes to more complex well-tuned edit distances, see for example, [11,13,37,38].

3. Contributions

We propose a new generalized domain-independent framework and subsequently three classes of algorithms for detecting near-duplicates among entities within one or more attributes in large database sets. These algorithms are synthetically complemented by near-duplicate generator algorithm (NDG). In the rest of the paper, we refer to such a framework as Merge-Filter Representative-based Clustering (Merge−Filter−RC) and to such a set of algorithms as constant threshold (*CT*), variable threshold (*VT*), and function thresholds (*FT*), respectively. For convenience, we collectively refer to these three algorithms as All−Three algorithms and to a specific algorithm by its conventional name, either *CT*, *VT*, or *FT*. In addition, we integrate an efficient synthetic near-duplicate generator algorithm (*NDG*) into All−Three algorithms. The NDG algorithm is capable of scaling up to generate algorithmically tens of millions of synthetic records. One of our aim here is to capture and modify the full power of Monge-Elkan (*ME*) [4,14] and Smith-Waterman (*SW*) [37] algorithms in the context of our work. That is, Merge−Filter−RC is a domain-independent approach that combines the token-based of Monge-Elkan and the character-based internal of Smith-Waterman similarity function, both of which we modified and augmented with a well-tuned affine parameterization. Still, this is not enough to solve the quality and complexity of our approach due to

anomalies which are associated with the transitivity relation and threshold choices. Thus, we segregate the results of the comparisons by labels which we refer to as *very similar*, *similar*, or *not similar*, and furthermore minimize an objective function without the user's intervention. Each constructed cluster has one or more *representatives* which are dynamically computed to measure the prototypicality of each record in the cluster. The record comparison is performed with only representatives rather than with all records in the cluster. Thus, records do not have to be compared to all other records, but to only cluster representatives which are considered for subsequent comparisons. We introduce cluster representatives which retain the most relevant syntactic and semantics features of the records in the cluster. The idea behind this approach is that cluster representatives reduce the total number of record comparisons, without reducing substantially the accuracy of the duplicate detection process.

Clusters' representatives are dynamically distilled and accurately achieved throughout a set of comparison functions, very similar, similar, or not similar, and threshold settings, constant, variable, or function. The number of similarity comparisons is reduced from $O(n^2)$ to $O(nm)$, where m and and n are the number of representatives and records, respectively and $m \ll n$, m is always independent of n, but dependent on the class of the algorithm. Each of these algorithms has a different impact when running on a set of thresholds, a constant, variable, or function. All algorithms are implemented in C++ and use the same data set to make fair comparisons. We ran an extensive experimental study using several real benchmarks and algorithmically generated synthetic data. We do not assume any specific structure in the data nor rely on any information available in the source data. That is, data has not been standardized, preprocessed, nor transformed, and syntactic as well as semantic errors remain as potential errors in the data. Experimental implementations show the Merge−Filter−RC detection approach greatly reduces the number of comparisons and the precision achieves a value of nearly 1.0, a precision which is close to the optimal outperforming consistently the seminal work of Monge-Elkan.

Our proposed set of algorithms do not presume a specific application domain, but in contrast they are tuned toward any domain-independent applications. Monge-Elkan's (ME) algorithm is relatively domain-independent with the purpose of integrating and matching web scientific papers from multiple sources, typically an alphanumeric domain class. The parameters used in ME are mapped to such a class of applications with only a restricted possibility of tuning the threshold values to provide a better accuracy. In addition, the heuristic method of ME minimizes the number of pairwise record comparisons with potential record duplicates and integrates some key concepts such as the minimum edit-distance of *SW*.

The rest of the paper is organized as follows. Section 3 reviews and summarizes the effectiveness of Monge-Elkan and Smith-Waterman algorithms. Section 4 addresses the metric measures used in detecting near-duplicates. Section 5 explains how to calculate and choose between precision and recall using F-measure. The choice, adjustment, and threshold tuning are defined in this section.

Section 6 proposes the merge-filter cluster representative framework which addresses the details of the accuracy performance using cluster representatives throughout precision and recall metrics. Section 7 provides a fully algorithmic technique and presents three different domain-independent algorithms, constant, variable, and function thresholds for detecting near-duplicates, all of them under the umbrella of Merge−Filter−RC.

In Section 8, we present a thorough experimental evaluation of our approach and compare the effectiveness of All−Three algorithms in terms of accuracy and efficiency with the seminal work of Monge-Elkan. We used benchmark data as well as synthetic data to meet specific conditions that are not available in existing real data. Synthetic data has been generated using the near-duplicate generator (NDG) algorithm. Section 9 concludes the paper with potential and future research directions. In addition, we provide an appendix to make our paper self contained.

4. Effectiveness of Monge-Elkan and Smith-Waterman algorithms

In this paper, we focus on the Smith-Waterman (*SW*) edit distance [37] that was originally developed for identifying common molecular subsequences, like DNA or proteins. Two strings x and y may not be entirely similar but contain regions, perhaps in the middle, that exhibit high similarity. Finding such a pair of regions, one from each of the two strings, is referred to as a *local alignment*. The Smith-Waterman algorithm finds the local alignment between two strings with the maximum possible score using a dynamic approach that runs in $O(|x||y|)$ time. The main limitation of SW is it places heavier penalties (*i.e.,* higher cost) on mismatches in the middle of strings rather than at the beginning and the end of strings. This may create a problem when the errors are in the middle of the strings. In this regard and in order to eliminate this inconvenience, we primarily consider the Monge-Elkan's methodology [14,26], a well-tuned matching methodology normalized in the interval $[0, 1]$, allowing additional parameters, and introduces gaps in the alignment of two strings. Much of the power of the Monge-Elkan's algorithm is due to its ability to include sequences of non-matching characters, gaps (*affine gaps*), in the alignment of two strings. In our work, we add the gap cost as another variant of the Smith-Waterman (*SW*) algorithm that offers a solution to the above problem and other related duplicate detection issues. By adding a cost, we extend the two extra edit operations, *starting gap* and *extending gap*.

In general and for instance, the gap penalty denoted by $\text{cost(gap)} = s + e \times l$, where s is the affine cost of starting a gap in an alignment, e is the cost of extending a gap, and l is the length of a gap in the alignment of two strings. Usually affine gap penalizes gap extension less than gap opening ($e < s$) thus we decrease the penalty for contiguous mismatched substrings by using a single long gap over many short gaps. Since the differences between near-duplicate records often arise because of many abbreviations or extra-string insertions and omissions, the affine-gap model produces a better similarity and more accurate results than most the other edit distance metrics. Moreover, the affine-gap algorithm performs well to detect similarities when records have minor syntactical differences, including typographical errors, abbreviations, and truncations. In fact, the Monge-Elkan's algorithm approximates the solution to the optimal assignment problem in combinatorial optimization. This approximation is a reasonable trade-off between accuracy and complexity.

In sum, Monge-Elkan's complexity is quadratic in number of tokens and one can define the Monge-Elkan's measure over two text strings that contain several tokens as:

$$MongeElkan(x, y) = \frac{1}{|x|}\sum_{i=1}^{|x|} \underset{j=1,|y|}{max}\ sim(x[i], y[i])$$

where $|x|$ and $|y|$ are the number of tokens in x and y, respectively and $sim(x, y)$ is an internal similarity function to measure the similarity between two individual tokens.

In the paper, we adopt a modified version of the Smith-Waterman similarity edit distance as inter-token similarity measure. Formally, let $c(x_i, y_i)$ denote the cost of the edit distance that aligns ith character of string x to j^{th} character of string y. Then the SW algorithm computes a cost matrix M that represents a maximum-cost string alignment by the following recurrence rule based on Monge-Elkan's algorithm [4].

$$M(i,j) = max\begin{cases} M(i-1,j-1)+c(x_i,y_j) & \\ M(i-1,j)+e & \text{if align}\,(i-1,j-1)\,\text{ends in a gap} \\ M(i-1,j)+s & \text{if align}\,(i-1,j-1)\,\text{ends in a match} \\ M(i,j-1)+e & \text{if align}\,(i-1,j-1)\,\text{ends in a gap} \\ M(i,j-1)+s & \text{if align}\,(i-1,j-1)\,\text{ends in a match} \end{cases}$$

5. Metric measures and threshold effects

Similarity refers to a measure of likeness between two objects (*i.e.,*

records, entities); and the dissimilarity between two objects is refereed to as a distance. We define a record as a set of tokens drawn from a finite universe $\mathcal{U}$. Let n be the number of real-world entities over a plurality of large databases that consist of records $\mathsf{R}_n = r_1, r_2, \ldots, r_n$, where a large number of r_i are potential near-duplicates. We denote by $\mathcal{D} = \mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_d$ the set of problem domains.

Similarity and distance measures are often normalized in the range $[0, 1]$, and $[0, \infty]$ or $[0, \text{some distance}]$, respectively. Formally, we define a similarity and distance measures as follows:

Definition 5.1. A similarity measure is a non-negative function sim: $\mathcal{D}_1 \times \mathcal{D}_2 \rightarrow [0,1]$, such that $\text{sim}(r_1, r_2) = 0$ if r_1 and r_2 are least similar and $\text{sim}(r_1, r_2) = 1$ if r_1 and r_2 are identical. ($\mathcal{D}_1$ might be equal to $\mathcal{D}_2$).

Definition 5.2. A distance measure is a non-negative function dist: $\mathcal{D}_1 \times \mathcal{D}_2 \rightarrow [0, 1]$, such that $\text{dist}(r_1, r_2) = 0$ if r_1 and r_2 are exactly similar or identical, and $\text{dist}(r_1, r_2) = 1$ if r_1 and r_2 are not similar.

The search space for detecting near-duplicates can be reduced under the assumption the relation "*is duplicate of*" or "*is similar to*", is transitive. However, the theory of transitivity is not always flawless in practice because of the propagation of errors as explained earlier. Duplicate records tend to be sparsely distributed over a large database space and the propagation of errors is statistically insignificant [26]. The complexity and evaluation measures to assess a near-duplicate record algorithm that have been addressed are the *efficiency* and *accuracy*. Accuracy is considered the most important quality assessment dimension in near-duplicate algorithms, and it is measured in terms of two prominent measures, *precision* and *recall*. The other related measure is F-measure which determines the harmonic mean of the precision and recall values. The goal of our evaluation is to find the best metrics in terms of quality and effectiveness with respect to different domain-independent data sets. We classify each value pair of comparisons as *very similar*, *similar*, or *not similar* since errors may occur during the process of detecting near-duplicates. For completeness, as it is illustrated in Fig. 1, we divide the search space into subspaces and denote by true positives all candidate pairs that are correctly declared to be duplicates (*i.e.,* expected matches), and false positives all candidate pairs that are incorrectly declared to be duplicates while in fact they may not be duplicates (*i.e.,* there should not be a match).

Similarly, true negatives are pairs that are correctly recognized as not being duplicates (*i.e.,* expected mismatch), and false negatives are pairs that are not declared to be duplicates while in fact they are (*i.e.,* there should be a match). The metric precision measures the fraction of correct duplicates over the total number of record pairs classified as duplicates by the algorithm. The metric recall measures the fraction of records correctly clustered over the total number of duplicates. A high recall means no false misses and a high precision means few false matches; thus, there is a trade-off between high recall and high precision. The other measure we want to introduce is the *reduction ratio*, which is the relative reduction in the number of pairs to be compared. This means a search space reduction strategy is needed in order to reduce the number of record comparisons. As a consequence, we introduce *prototype cluster representatives* which retain the most relevant syntactic and semantics features of the records in the cluster where comparisons take place with cluster representatives, instead of all records, thus the search space can be reduced with the improvement of both true and declared subspaces. In other words, the reduction of false positives and in particular false negatives have an impact on the accuracy. A high recall means no false misses and indicates high accuracy of the duplicate detection results. A high reduction ratio achieves an even more effective search space reduction. A high precision means few false matches and has the opposite effect than recall. The other metric measure, reduction ratio in the range of $[0, 1]$, is defined as 1 - (declared duplicates/all tuple pairs). The similarity threshold line in Fig. 1 ensures a trade-off among recall, precision and reduction ratio. If we shift the similarity threshold line (similarity threshold 1) to the right, consequently more tuple pairs will be rejected and increase the reduction ratio and precision. This rejection would ultimately lead to decrease the recall metric. If we shift the similarity threshold line to the left then the opposite trade-off effect between recall, precision, and reduction ratio will take place. Similarly and in the same fashion, shifting the similarity threshold line (similarity threshold 2) would have an effect on true and false positives.

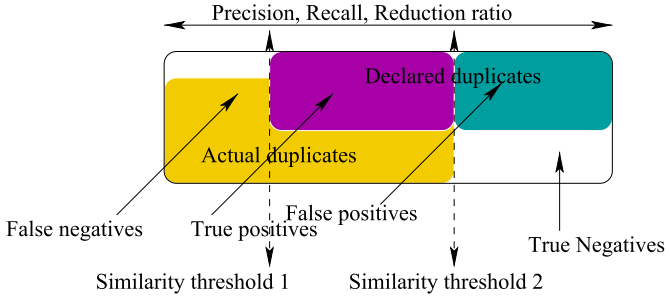


Fig. 1. Similarity threshold factors and tradeoff between prominent measure metrics.

Relations between threshold similarity metrics and accuracy are an important part of the near-duplication detection process. Different threshold setting and parameter tuning (*i.e.,* distance functions, window size) have been used to classify whether a pair of records is a duplicate or a non-duplicate. Several experiments have been conducted and cutoff thresholds for a specific application domain with its own characteristics, often manually configured, have shown to achieve the "best" precision, recall and F-measure values. The cutoff value of the threshold along with other key tuning parameters can be very time consuming as the search space can grow exhaustively and even exponentially, which necessitates some forms of threshold optimization [35,39,40].

6. Merge-Filter Representative-based Clustering: A near-optimal domain-independent approach

Similar in spirit to divide and merge methodology for clustering [41], Merge-Filter Representative-based Clustering (*Merge−Filter−RC*) combines a top-down divide phase with a bottom-up merge phase in a hierarchical scheme as described in subsequent sections. Merge−Filter−RC is made of two trees, a top-down and bottom-up tree, annotated with cluster records. Merge−Filter−RC is mainly geared towards the needs of detecting near-duplicates with a provision of forming the best possible near-optimal clusters of records by integrating a variant of Smith-Waterman's and Monge-Elkan's algorithms into the detection approach. Importantly, we are only interested in unconstrained algorithms, rather than adopting any standard clustering algorithm. That is, there is no preliminary assumption of rationality to choose the number of clusters as input or other domain specific parameters. Let $\mathbb{C}$ denote the initial cluster containing n records in R_n, and d denote the depth of a node in the top-down cluster tree T whose root $(\mathbb{C}, n)$ is at $d = 0$ and internal nodes are $(\mathscr{C}_i, nb)$, where C_i and nb refer to the name of the cluster and the number of records in $\mathscr{C}_i$, respectively as illustrated in Fig. 2. The divide phase of Merge−Filter−RC recursively splits the collection of records into two equal halves at each level of T and constructs the tree T on the basis of these records. The near-duplicate detection process starts with the data set of the initial cluster $\mathbb{C}$ that is expected to contain near-duplicate records. We start the construction at the root with n records and end-up at the leaf level with one single record per cluster, $(\mathscr{C}_l, 1)$ (Fig. 2). The number of times the split is done is exactly equal to the height of the tree T, $O(\log n)$, because the size of the clusters decreases approximately by half at each level of T which is defined as follows:

Definition 6.1. A top-down cluster tree T is a full binary tree such that (i) the nodes of T are subclusters of $\mathbb{C}$ (ii) every internal node of T has two subcluster of records, each of size $n/2^d$, (iii) every leaf of T is a subcluster of records of size one; (iv) the root node of T is the cluster $\mathbb{C}$ that consists

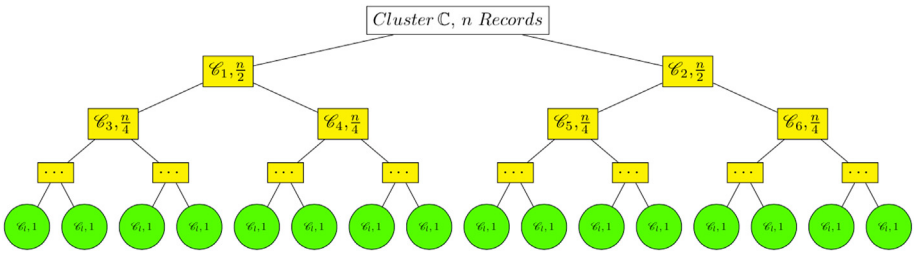


Fig. 2. Construction of a top-down cluster tree T.

of n records.

Let $\mathbb{C} = \mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ be the set of clusters of T produced by the divide phase. Each internal node of T is a subset of the data set records. The left and right children of a node $\mathscr{C}_i$ forms a partition of the parent such that $|\mathscr{C}_i| = n/2^d$ and $1 \leq i, i \leq n$ (Fig. 2). Let T be a top-down cluster tree then for any two clusters we have either $\mathscr{C}_i \subset \mathscr{C}_j$ or $\mathscr{C}_j \subset \mathscr{C}_i$ or $\mathscr{C}_i \cap \mathscr{C}_j = \emptyset, i \neq j, 1 \leq i, j \leq n$.

Starting at the leaves of T, the bottom-up merging phase is applied recursively to each node of T towards the root enabling the construction of a new tree of clusters whose root is $\hat{T}$ as depicted in Fig. 3. To accomplish this, the near-optimal cluster of an interior node $\hat{\mathscr{C}}_i$ in the tree $\hat{T}$ is obtained by merging and distilling dynamically the near-optimal clusters of the left and right children of $\hat{\mathscr{C}}_i$. The result of clustering T is a partition $\hat{\mathbb{C}} = \hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_q$ where one or more $\hat{\mathscr{C}}_i$ are the nodes of $\hat{T}$, referred to as near-optimal duplicate tree, $i \leq q$ and $q \ll n$ (q is much less than n). The value of q is not known in advance and is independent of n.

At the beginning of the Merge−Filter−RC bottom-up phase, we initialize the set of leaf clusters of $\hat{T}$ with the set of leaf clusters of T. Then, we apply a hierarchical agglomerative clustering to the leaf clusters by bringing the leaves up to the root level by level, and near-optimizing the objective function locally at each iteration when two clusters are compared and eventually merged. The choice of the objective function uses the dynamic programming of SW algorithm refined by different parameters as it will be explained in the next sections. Each constructed cluster $\hat{\mathscr{C}}_i$ has one or more cluster representatives that are dynamically computed to measure the prototypicality of each record in the cluster. The record comparison is performed with only representatives rather than with all records in the cluster. Consequently, records do not have to be compared to all others but only cluster representatives are considered for subsequent comparisons. That is, if a given record is not similar to a record(s) in a set of cluster representatives then it will not match the other record members of the cluster. In general, Merge−Filter−RC takes as parameters two unordered pairs of clusters ($\hat{\mathscr{C}}_i$, $\hat{\mathscr{C}}_j$), and their respective cluster representatives ($\hat{\mathcal{R}}_i$, $\hat{\mathcal{R}}_j$) where $1 \leq i, j \leq n$, then returns whether ($\hat{\mathscr{C}}_i$, $\hat{\mathscr{C}}_j$) are *very similar*, *similar* or *not similar* by finding the optimal local alignment using the SW algorithm with the maximum similarity score. Let such a set of cluster representatives denoted by $\hat{\mathcal{R}}_i = \{\hat{r}_{i1}, \hat{r}_{i2}, \ldots, \hat{r}_{ij}\}$, $1 \leq j \leq l$ where l indicates the number of representatives for a fixed i. $\left|\hat{\mathscr{C}}_i\right| = k$ and $\left|\hat{\mathcal{R}}_i\right| = l$, where l is much less than k. Every generated ith cluster $\hat{\mathscr{C}}_i$, represented as a node in $\hat{T}$, has a set of near-duplicates and cluster representatives referred to as $\{r_{ij}\}_{j=1}^{k}$ and $\{\hat{r}_{ij}\}_{j=1}^{l}$, respectively. The results of detecting near-duplicates, however, may become sensitive to the initial selection of representatives. Initially, Merge−Filter−RC uses one record as a cluster representative, then representative(s) might be subsequently updated, either by retaining the same ones or iteratively re-computing and re-assigning new representatives. In addition, Merge−Filter−RC enforces transitivity between records in a cluster $\hat{\mathscr{C}}_i$. Each remaining record is compared to the representatives and placed in the cluster of the closest representative. The idea behind this approach is that cluster representatives reduce the total number of record comparisons, without reducing substantially the accuracy of the duplicate detection process. The number of similarity comparisons to be considered is reduced from $O(n^2)$ to $O(nm)$, where m and and n are the number of representatives and records, respectively. $m \ll n$ and m is always independent of n, but dependent on the class of the algorithm (constant, variable, or function).

The cornerstone property of Merge−Filter−RC is based on the choice of the appropriate objective function g and its parameters. We are interested in finding locally the optimal clustering at each level of $\hat{T}$, and also finding globally the near-optimal clustering at the root created by the Merge−Filter−RC merge phase. That is, g should guarantee to find the optimal local alignment, $\mathsf{OPT}(\mathcal{A})$, and quantifies the similarity in terms of a high-scoring alignment $\mathcal{A}$. That is, we maximize the objective function g by assigning a score for each alignment obtained by the SW algorithm. Let Σ^* denote the set of finite words over an alphabet Σ and $x = x_1 \ldots x_p, y = y_1 \ldots y_q$ where $x, y \in \Sigma^*$. The state space of all alignments of x and y is the mapping g: $(\langle x, y \rangle) \mapsto \mathbb{Z}$, the set of integers. The optimal local alignment score of the subsequences $x_1 \ldots x_p$ and $y_1 \ldots y_q$ is obtained by maximizing g among all alignments. That is,

$$g_0(\langle x, y \rangle) = \max_{\mathcal{A}} \; g(\langle x, y \rangle)$$

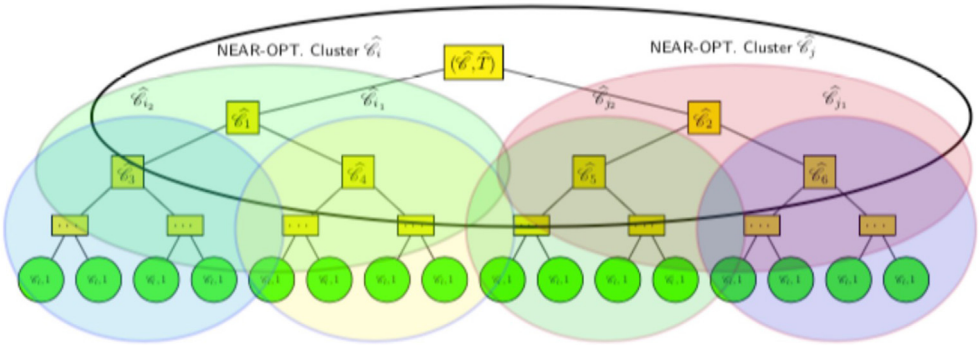


Fig. 3. Vizualization of a near-optimal duplicate cluster tree $\hat{T}$.

$$\mathrm{OPT}(\mathcal{A}) = \underset{\mathcal{A}}{argmax} g(\langle x, y\rangle)$$

Let.

$\hat{\mathscr{C}}_l$ and $\hat{\mathscr{C}}_r$ be the left and right children of an internal node $\hat{\mathscr{C}}$ in $\hat{T}$. Let $\mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}(\hat{\mathscr{C}}, i)$ be the near-optimal duplicate sub-tree for the node $\hat{\mathscr{C}}$ using i clusters as stated recurrently in the following theorem. The space of near-optical solutions is represented in a data structure. That is, a tree that can be efficiently used to find near-optimal solutions that satisfy the optimal local alignment. Near-optimal and recursively constructed bottom-up subtrees, *DUPTREE*($\hat{\mathscr{C}}$, i), facilitate the re-optimization (*i.e.,* tuning) in the object function g to satisfy the properties and accomplish the near-optimality solution. Thus, each subtree of the node $\hat{\mathscr{C}}$ using i clusters, *DUPTREE*($\hat{\mathscr{C}}$,i), transparently represents the space of near-optimal solutions and how each subtree relates to each other. We propose the following theorem which formally and compactly describes the near-optimal detection solution along with the corresponding objective function g. We explore the clustering methodology of [41] further and formally introduce near-optimal duplication tree of a node $\hat{\mathscr{C}}$ using i cluster defined as $\mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}(\hat{\mathscr{C}}, i)$. The objective function values of each alignment obtained by the SW algorithm guarantee the optimal local alignment across all subtrees of $\hat{T}$.

Theorem 6.1. $\mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}(\hat{\mathscr{C}}, i) =$

$$\begin{cases} \hat{\mathscr{C}} & \text{if } i=1 \\ \mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_l, j\right) \cup \mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_r, i-j\right) & \text{if } i>1 \end{cases}$$

where

$$j = \underset{1\le j<i}{argmin}\, g\left(\mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_l, j\right) \cup \mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_r, i-j\right)\right)$$

Proof. We proceed by induction on i. The base case handles all clusters of $\hat{T}$. That is, all initial clusters $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_q$ with a single representative that is generated in the divide phase. Starting at the leaves, the first optimal clusters are originated from the SW algorithm. If two records are in the same cluster then they are considered to be near duplicate or similar, and if not they are dissimilar.

For the induction case, we can now assume the claim is true for all i, 1 < i < n. Let inode($\hat{T}$) be an internal node of $\hat{T}$, and $\hat{T}_1$ and $\hat{T}_2$ the left and right subtrees recursively build from the leaf clusters of $\hat{T}$, respectively. Without loss of generality, define $\hat{\mathscr{C}}_1{}^l$ and $\hat{\mathscr{C}}_1{}^r$ to be the clusters whose root is $\hat{T}_1$; and $\hat{\mathscr{C}}_2{}^l$ and $\hat{\mathscr{C}}_2{}^r$ be the clusters rooted in the right subtree $\hat{T}_2$, ordered as $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_1{}^r), (\hat{\mathscr{C}}_2{}^l, \hat{\mathscr{C}}_2{}^r)$. Denote by $(\hat{\mathcal{R}}_1{}^l, \hat{\mathcal{R}}_1{}^r)$ and $(\hat{\mathcal{R}}_2{}^l, \hat{\mathcal{R}}_2{}^r)$, the set of cluster representatives of $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_1{}^r)$ and $(\hat{\mathscr{C}}_2{}^l, \hat{\mathscr{C}}_2{}^r)$, respectively. Let $\hat{\mathcal{R}}_1{}^l = \hat{\mathcal{R}}_1{}^l \cup \hat{\mathcal{R}}_1{}^r$ and $\hat{\mathcal{R}}_2{}^r = \hat{\mathcal{R}}_2{}^l \cup \hat{\mathcal{R}}_2{}^r$ such that $\left|\hat{\mathcal{R}}_i{}^l\right| = p > 1$ and $\left|\hat{\mathcal{R}}_i{}^r\right| = q > 1$, for i = 1, 2. That is, a cluster has more than one representative which is used for subsequent comparisons. Importantly, representative clustering reduces the total number of record comparisons substantially and furthermore representatives are bounded by two similarity threshold values, θ_u and θ_l. Keeping this potential of multiple cluster representatives, $\hat{\mathscr{C}}_l = \hat{\mathscr{C}}_1{}^l \cup \hat{\mathscr{C}}_1{}^r$ and $\hat{\mathscr{C}}_r = \hat{\mathscr{C}}_2{}^l \cup \hat{\mathscr{C}}_2{}^r$. For convenience, let $\hat{\mathcal{R}}_1{}^l = \{\hat{r}^l_{1,1}, \hat{r}^l_{1,2}, \ldots, \hat{r}^l_{1,p}\}$ and $\hat{\mathcal{R}}_2{}^l = \{\hat{r}^l_{2,1}, \hat{r}^l_{2,2}, \ldots, \hat{r}^l_{2,p}\}$. Similarly $\hat{\mathcal{R}}_1{}^r$ and $\hat{\mathcal{R}}_2{}^r$ are defined. If two records (i.e., representatives) are in the same cluster then they are considered to be near-duplicates or exactly similar, and if not they are dissimilar. For instance, let $\mathrm{R}_3 = \{r_1, r_2, r_3\}$ be the set of records to be compared and the initial good threshold chosen is in the interval [0.83⋯0.90]. Assume we have the following similarities: sim(r_1, r_2) = 0.88, sim(r_1, r_3) = 0.72, and sim(r_2, r_3) = 0.80. Then, (r_1, r_2) are classified as near-duplicates and assigned to the same cluster. However, (r_1, r_3) and (r_2, r_3) would be non-duplicates. Now, consider a new record r_4 added to R_3 with sim(r_1, r_4) = 0.75, sim(r_2, r_4) = 0.88, and sim(r_3, r_4) = 0.96. Then, (r_2, r_4) and (r_1, r_4) are classified as near-duplicated and non-duplicated, respectively. However, due to the anomaly in the transitivity relation and the non-appropriate choice of the threshold, the pair of records, (r_1, r_4) and (r_2, r_3), would be reclassified as near-duplicates although there were not. This is an important note that should largely foster the choice of appropriate thresholds for larger data sets. With our proposed approach, the results of the comparisons are segregated by labels referred to as very similar, similar, or not similar; and the quality and accuracy of the classification is further improved by finding near-optimal or sub-optimal thresholds to identify more accurately duplicate records by minimizing the objective function without the users intervention. With this viewpoint and based on the SW algorithm, we start computing the optimal clustering for the leaf nodes and then find the near-optimal clustering, relatively to the optimal local alignment, for any internal node. That is, at the end of the SW algorithm during the merge phase, $\mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}(\hat{T})$ gives the almost optimal clustering. We consider two upper and lower bound threshold values, θ_l and θ_u, which are extensively studied in domain independent large databases. Both bounds which are rooted in the seminal Monge-Elkan's algorithm have shown effectiveness as they become core standard thresholds in research literature [39,40,42] and in almost every approximate duplicate detection algorithm. The two upper and lower bound threshold values are mainly governed by the probability of errors for finding optimal record alignments. Thus, we set the semantic similarity threshold parameter for which two records are considered semantically similar to θ_u. In the same way as semantically similar records, we set the non-similarity threshold to θ_l.

We set θ_u to 0.7 and θ_l to 0.5 for declaring two records as near-duplicate and non-duplicate, respectively. $\theta \le \theta_l$ produces loose similarity (not similar), $\theta \ge \theta_u$ produces high similarity (very similar), and if θ falls between these values it is regarded as a regular similarity (similar). Let $\hat{\mathscr{C}}_l$ and $\hat{\mathscr{C}}_r$ be the left and right children of an internal node $\hat{\mathscr{C}}$ whose cluster representative is $\hat{\mathcal{R}}$. We denote by $\hat{\mathcal{R}}_l$ and $\hat{\mathcal{R}}_r$ the left and right cluster representative children of $\hat{\mathcal{R}}$. As a consequence of Theorem 6.1, we state the following result.

Corollary 6.1. $\mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}(\hat{\mathcal{R}}, i) =$

$$\begin{cases} \hat{\mathcal{R}} & \text{if } i=1 \\ \mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathcal{R}}_l, j\right) \cup \mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathcal{R}}_r, i-j\right) & \text{if } i>1 \end{cases}$$

where

$$j = \underset{1\le j<i}{argmin}\, g\left(\mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathcal{R}}_l, j\right) \cup \mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathcal{R}}_r, i-j\right)\right)$$

7. Competitive algorithms

We introduce three classes of algorithms from different perspectives, a constant threshold (CT), a variable threshold (VT), and a function threshold (FT) algorithm, collectively referred to as All−Three algorithms. Each of these algorithms has a different impact when running on various benchmarks and synthetic data sets.

7.1. Constant threshold (CT) algorithm

We use the same notations in line with Section 6. Let $\hat{T}$ be the bottom-up cluster tree, and $\hat{T}_1$ and $\hat{T}_2$ the left and right subtrees recursively build from the leaf clusters of $\hat{T}$, respectively. Assume n, the number of records, is a power of 2 for the sake of convenience. The output of the approximate duplicate SW merge phase is a partition of clusters $\hat{\mathbb{C}} = \hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots,$

$\hat{\mathscr{C}}_q$, where each $\hat{\mathscr{C}}_i$ is a node of $\hat{T}$ and q is unknown in advance. Without loss of generality, let assume $\hat{\mathscr{C}}_1{}^l$ and $\hat{\mathscr{C}}_1{}^r$ be the clusters whose root $\hat{T}_1$; and $\hat{\mathscr{C}}_2{}^l$ and $\hat{\mathscr{C}}_2{}^r$ be the clusters rooted in the right subtree $\hat{T}_2$ of $\hat{T}$, ordered as $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_1{}^r), (\hat{\mathscr{C}}_2{}^l, \hat{\mathscr{C}}_2{}^r)$.

The procedure Constant Threshold $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$ in algorithm 1 is based on the idea of the Merge−Filter−RC approach which compares and subsequently updates two given clusters, either by merging them into a new cluster and removing one of the original cluster. Moreover, the algorithm iteratively recomputes and reassigns new representatives. Each record is compared to the representatives and placed in the cluster of the closest representative. The number of comparisons is reduced from $O(n^2)$ to $O(mn)$, where reduced m and n are the number of representatives and records, respectively (m is much smaller than n).

7.2. Variable threshold (VT) algorithm

In procedure Variable Threshold $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$ in algorithm 1, each cluster has only one representative and only one variable threshold value is considered, starting at $\theta_1 = 0.5$. In line with the approximate duplicate SW algorithm, the variable threshold algorithm (VT) compares clusters from the left subtree with clusters from right subtree. That is, we compare $\hat{\mathscr{C}}_1{}^l$ with $\hat{\mathscr{C}}_2{}^l$ and $\hat{\mathscr{C}}_2{}^r$, then we compare $\hat{\mathscr{C}}_1{}^r$ with $\hat{\mathscr{C}}_2{}^l$ and $\hat{\mathscr{C}}_2{}^r$. Without loss of generality and for algorithmic simplicity, we assume cluster $\hat{\mathscr{C}}_1$ has $\hat{r}_1$ as a record representative and θ_1 as a threshold. Similarity, $\hat{\mathscr{C}}_2$ has $\hat{r}_2$ as a record representative and θ_2 as a threshold. At start, clusters have one record and the threshold value θ_1 is set to 0.5.

7.3. Function threshold (FT) algorithm

In this third category of algorithms each cluster has more than one representative. Furthermore, two variable threshold values are considered, an upper bound value $\theta_u = 0.7$, and a calculated threshold value θ_c. In line with the approximate duplicate SW algorithm, the function threshold algorithm (FT) compares clusters from the left subtree with clusters from right subtree. That is, we compare $\hat{\mathscr{C}}_1{}^l$ with $\hat{\mathscr{C}}_2{}^l$ and $\hat{\mathscr{C}}_2{}^r$, then we compare $\hat{\mathscr{C}}_1{}^r$ with $\hat{\mathscr{C}}_2{}^l$ and $\hat{\mathscr{C}}_2{}^r$. In addition, we compare $\hat{\mathscr{C}}_1{}^r$ with every cluster in the second half subtree.

The function Compare $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$ shows the very similar case when max SW $> \theta_u$ (lines 4 and 5). The other two cases, similar ($\theta_l \leq$ max SW $\leq \theta_u$) and not similar (max SW $< \theta_l$), can also be treated in a same manner. Lines 4 and 5 in the function Compare $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$ should be substituted by lines 4 and 5 in the function Check-Similarity $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$, respectively. In the same way, lines 11 and 12 should be substituted by lines 4 and 5. For the case of no similarity, lines 4 and 5 in the function Compare $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$ should be substituted by lines 7 and 8 in the function Check-Similarity $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$.

In the same way lines 11 and 12 should be substituted by lines 7 and 8. Let $\hat{\mathcal{R}}_1$ and $\hat{\mathcal{R}}_2$ be the set of the cluster representatives of $\hat{\mathscr{C}}_1$ and $\hat{\mathscr{C}}_2$, respectively such that $\left|\hat{\mathcal{R}}_1\right| = p$ and $\left|\hat{\mathcal{R}}_2\right| = q$, where $p, q \geq 1$. For algorithmic convenience, let assume that $q \geq p$. Let $\hat{\mathcal{R}}_1 = \{\hat{r}_{1i}\}_{i=1}^{p}$ and $\hat{\mathcal{R}}_2 = \{\hat{r}_{2j}\}_{j=1}^{q}$ be the cluster representatives of $\hat{\mathscr{C}}_1$ and $\hat{\mathscr{C}}_2$, respectively. The value max SW refers to the value returned by the Smith-Watermann algorithm.

Algorithm 1
All − Three Algorithms.

1: **Initialization:** Two cluster leaf records $(\mathscr{C}_l, \mathscr{C}_l)$
2: **procedure** Constant Threshold $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$ \(▷\) CT algorithm

(*continued on next column*)

Algorithm 1 (*continued*)

3: Compare $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$ \(▷\) comparison of $\hat{\mathscr{C}}_1{}^l$ with $\hat{\mathscr{C}}_2{}^l$
4: **if** $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$ are very similar **then**
5: $\hat{\mathscr{C}}_{new} \leftarrow$ Merge $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$
6: Remove $\hat{\mathscr{C}}_2{}^l$
7: $\hat{\mathcal{R}}_{new} \leftarrow \hat{\mathcal{R}}_1{}^l$ \(▷\) cluster rep. ← rep. of $\hat{\mathscr{C}}_1{}^l$
8: **end if**
9: **if** $\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l$ are similar **then**
10: $\hat{\mathcal{R}}_{new} \leftarrow$ Merge $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$
11: Remove $\hat{\mathscr{C}}_2{}^l$
12: $\hat{\mathcal{R}}_{new} \leftarrow \hat{\mathcal{R}}_1{}^l \cup \hat{\mathcal{R}}_2{}^l$ \(▷\) cluster rep. ← rep. of $\hat{\mathscr{C}}_1{}^l \cup$ rep. of $\hat{\mathscr{C}}_2{}^l$
13: **end if**
14: **if** $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$ are not similar **then**
15: no operation
16: **end if**
17: Goto step 1 and compare $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^r)$
18: Goto step 1 and compare $(\hat{\mathscr{C}}_1{}^r, \hat{\mathscr{C}}_2{}^l)$
19: Goto step 1 and compare $(\hat{\mathscr{C}}_1{}^r, \hat{\mathscr{C}}_2{}^r)$
20: **return** $(\hat{\mathcal{R}}_1{}^l, \hat{\mathcal{R}}_2{}^l)$\(▷\) returns cluster representatives
21: **end procedure**

1: **procedure** Variable Threshold $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$ \(▷\) VT algorithm
2: Compare and find SW of $\hat{r}_1$ and $\hat{r}_2$ \(▷\) at start θ_1, θ_2 are 0.5 for $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2$
3: **if** (SW $> \theta_1$ or (SW $> \theta_2$) **then**
4: **if** $(\theta_1 > \theta_2)$ **then**
5: Merge $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$
6: Remove $\hat{\mathscr{C}}_2$
7: Set representative ← $\hat{r}_1$ \(▷\) update representative to $\hat{r}_1$
8: $\theta_1 \leftarrow (\theta_1 +SW)/2$ \(▷\) update θ_1
9: **end if**
10: **if** $(\theta_2 \geq \theta_1)$ **then**
11: Merge $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$
12: Remove $\hat{\mathscr{C}}_1$
13: Set representative ← $\hat{r}_2$ \(▷\) update representative to $\hat{r}_2$
14: $\theta_2 \leftarrow (\theta_2 +$ SW$)/2$
15: **end if**
16: **else**
17: $\hat{\mathscr{C}}_1$ and $\hat{\mathscr{C}}_2$ are not similar
18: no operation
19: **end if**
20: **end procedure**

1: **procedure** Function Threshold $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^l)$ \(▷\) FT algorithm
2: Find max SW between $\hat{\mathscr{C}}_1{}^l$ and every cluster in $\hat{T}_2$
3: Suppose maxSW is between $\hat{\mathscr{C}}_1{}^l$ and $\hat{\mathscr{C}}_2{}^r$
4: **if** (max SW $> \theta_u$) **then**
5: Merge $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^r)$
6: Remove $\hat{\mathscr{C}}_1{}^l$
7: Representative ← representative of $\hat{\mathscr{C}}_2{}^r$
8: **end if**
9: **if** $(\theta_c \leq$ max SW $< \theta_u)$ **then**
10: Merge $(\hat{\mathscr{C}}_1{}^l, \hat{\mathscr{C}}_2{}^r)$
11: Remove $\hat{\mathscr{C}}_2{}^r$
12: Representative ← Union of representatives
13: **end if**
14: **if** (max SW < COMPUTE(θ_c) **then**
15: no operation
16: **end if**
17: **end procedure**

1: **function** Compare $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$ \(▷\) very similar case
2: Compare and find SW of $\hat{r}_{11}$ with each $\{\hat{r}_{2j}\}_{j=1}^{q}$
3: Find max SW; $m \leftarrow$ max SW
4: **if** at any time (max SW $> \theta_u$) **then**
5: $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2)$ are very similar
6: Stop and return max SW
7: **end if**
8: **if** (max SW $< \theta_u$) **then**
9: Compare and find SW of $\hat{r}_{12}$ with each $\{\hat{r}_{2j}\}_{j=1}^{q}$
10: Find maxSW; start with max SW equal to m
11: **if** at any time (max SW $> \theta_u$) **then**

(*continued on next page*)

Algorithm 1 (*continued*)

12: ($\hat{\mathscr{C}}_1$, $\hat{\mathscr{C}}_2$) are very similar
13: Stop and return max SW
14: **end if**
15: **end if**
16: **end function**

1: **function** COMPUTE (θ_c)
2: $t \leftarrow 0.3\ (1 - \theta_u)$ \(▷\) θ_u: Monge-Elkan's threshold
3: $x \leftarrow$ the number of representative in $\hat{\mathscr{C}}_2^{\,r} - 1$
4: Let $m \leftarrow$ minimum(x, maximum number of allowed representatives)
5: D $\leftarrow t$/(maximum number of allowed representatives)
6: $\theta_c \leftarrow 2 \times t - m \times$ D
7: **return** θ_c
8: **end function**

1: **function** CHECK SIMILARITY ($\hat{\mathscr{C}}_1$, $\hat{\mathscr{C}}_2$) \(▷\) Check degree of similarity: \(▷\) very similar, similar, not similar
2: **if** (max SW $> \theta_u$) **then**
3: ($\hat{\mathscr{C}}_1$, $\hat{\mathscr{C}}_2$) are very similar
4: **end if**
5: **if** (max SW $\leq \theta_u$) and (max SW $\geq \theta_l$) **then**
6: ($\hat{\mathscr{C}}_1$, $\hat{\mathscr{C}}_2$) are similar
7: **end if**
8: **if** (max SW $< \theta_l$) **then**
9: ($\hat{\mathscr{C}}_1$, $\hat{\mathscr{C}}_2$) are not similar
10: **end if**
11: **end function**

1: **function** COMPARE LEFT-RIGHT SUBTREES ($\hat{\mathscr{C}}_1^{\,l}$, ($\hat{\mathscr{C}}_2^{\,l}$, $\hat{\mathscr{C}}_2^{\,r}$))
2: Find max SW between $\hat{\mathscr{C}}_1^{\,l}$ and every cluster in $\hat{T}_2$ \(▷\) similar to function COMPARE
3: Suppose max SW is between $\hat{\mathscr{C}}_1^{\,l}$ and $\hat{\mathscr{C}}_2^{\,r}$
4: **if** (max SW $> \theta_u$) **then**
5: Merge ($\hat{\mathscr{C}}_1^{\,l}$, $\hat{\mathscr{C}}_2^{\,r}$)
6: Remove $\hat{\mathscr{C}}_1^{\,l}$
7: Representative $\leftarrow$ representative of $\hat{\mathscr{C}}_2^{\,r}$
8: **end if**
9: **if** ($\theta_c \leq$ max SW $< \theta_u$) **then**
10: Merge ($\hat{\mathscr{C}}_1^{\,l}$, $\hat{\mathscr{C}}_2^{\,r}$)
11: Remove $\hat{\mathscr{C}}_2^{\,r}$
12: Representative $\leftarrow$ Union of representatives
13: **end if**
14: **if** (max SW $< \theta_c$) **then**
15: do nothing
16: **end if**
17: **end function**

Without loss of generality, let assume $\hat{\mathscr{C}}_1^{\,l}$ and $\hat{\mathscr{C}}_1^{\,r}$ be the clusters whose root $\hat{T}_1$; and $\hat{\mathscr{C}}_2^{\,l}$ and $\hat{\mathscr{C}}_2^{\,r}$ be the clusters rooted in the right subtree $\hat{T}_2$ of $\hat{T}$, ordered as $(\hat{\mathscr{C}}_1^{\,l}, \hat{\mathscr{C}}_1^{\,r}), (\hat{\mathscr{C}}_2^{\,l}, \hat{\mathscr{C}}_2^{\,r})$.

As a result of Theorem 6.1, we state the following results:

Corollary 7.1. $\mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}(\hat{\mathscr{C}}_1^{\,l}, i) =$

$$\begin{cases} \hat{\mathscr{C}}_1^{\,l} & \text{if } i = 1 \\ \mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_2^{\,l}, j\right) \cup \mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_2^{\,r}, i-j\right) & \text{if } i > 1 \end{cases}$$

where

$$j = \underset{1 \leq j < i}{argming}\left(\mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_2^{\,l}, j\right) \cup \mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_2^{\,r}, i-\mathrm{j}\right)\right)$$

Corollary 7.2. $\mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}(\hat{\mathscr{C}}_1^{\,r}, i) =$

$$\begin{cases} \hat{\mathscr{C}}_1^{\,r} & \text{if } i = 1 \\ \mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_2^{\,l}, j\right) \cup \mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_2^{\,r}, i-j\right) & \text{if } i > 1 \end{cases}$$

where

$$j = \underset{1 \leq j < i}{argming}\left(\mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_2^{\,l}, j\right) \cup \mathrm{NEAR-OPT}_{\mathrm{DUPTREE}}\left(\hat{\mathscr{C}}_2^{\,r}, i-\mathrm{j}\right)\right)$$

8. Evaluation metrics and experiments

8.1. Evaluation metrics

The main goal of this experimental evaluation is to compare the performance in terms of accuracy and effectiveness of all four algorithms covered in this work, ME, CT, VT, and FT algorithms. We adopt the purity, inverse purity, and F-measure metrics for our extensive evaluation. Denote by $\hat{\mathbb{C}} = \hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_q$ the set of the true actual clusters where we refer to each $\hat{\mathscr{C}}_i$ as a class and let $\mathbb{C} = \mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ be the set of near-duplicate clusters to be evaluated by the SW algorithm. Then the precision, $P(\hat{\mathscr{C}}_i, \mathscr{C}_j)$ and recall $R(\hat{\mathscr{C}}_i, \mathscr{C}_j)$ of $\hat{\mathscr{C}}_i$ with respect to $\mathscr{C}_j$ are defined as follows:

$$P\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right) = \frac{\left|\hat{\mathscr{C}}_i \cap \mathscr{C}_j\right|}{\left|\mathscr{C}_j\right|} \quad \textit{and} \quad R\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right) = \frac{\left|\hat{\mathscr{C}}_i \cap \mathscr{C}_j\right|}{\left|\hat{\mathscr{C}}_i\right|}$$

where $P(\hat{\mathscr{C}}_i, \mathscr{C}_j) = R(\mathscr{C}_j, \hat{\mathscr{C}}_i)$. Let $\hat{n}$ be the total number of clustered entities, including near-duplicates, purity is formulated by the weighted average of the maximum precision values achieved by the clusters on one of $\hat{\mathscr{C}}_j$:

$$\{\mathrm{Purity}\} = \sum_{j=1}^{k} \frac{\left|\mathscr{C}_j\right|}{\hat{n}} \max_{i=1}^{q} P\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right)$$

However, inverse purity considers the cluster with maximum recall for each class $\hat{\mathscr{C}}_i$, and it is obtained by taking the weighted average of the maximum recall values

$$\textit{Inverse}\ \ \mathrm{Purity} = \sum_{i=1}^{q} \left|\hat{\mathscr{C}}_i\right| \hat{n} \max_{j=1}^{k} R\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right)$$

The values of the purity and inverse purity metrics are within the range of 0–1. The higher purity the better is the inverse purity. Purity penalizes clustering noise (i.e., duplicates) in a cluster, which means grouping records incorrectly, but it does not reward grouping records from the same $\hat{\mathscr{C}}_j$. If every cluster contains only one record, the maximum purity is 1. However, reversing the role of $\hat{\mathbb{C}}$ and $\mathbb{C}$, that is inverse purity ($\mathbb{C}, \hat{\mathbb{C}}$) would penalize splitting records belonging to the same cluster $\hat{\mathscr{C}}_j$ into different clusters. In other words, inverse purity rewards grouping near-duplicates together, but it does not penalize noisy records from different $\hat{\mathscr{C}}_j$. In a similar way to precision and recall, there is a tradeoff relationship between inverse purity and purity. Inverse purity rewards grouping near-duplicates and the maximum inverse purity is achieved by putting all records in one single cluster. For each class $\hat{\mathscr{C}}_i$, the F-measure of that class is:

$$F\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right) = \max_{j=1}^{k} \frac{2 \times P\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right) \times R\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right)}{P\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right) + R\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right)}$$

The F-measure, a combination of purity and inverse purity, is in the range of $[0, 1]$ and a higher F-measure indicates a better clustering. The F-measure, is in the range of $[0, 1]$, indicates a better clustering. The F-measure of the clustering [35,36] which computes the weighted average of maximal F-measure values is defined as:

$$F - measure = \sum_{i=1}^{q} F\left(\hat{\mathscr{C}}_i, \mathscr{C}_j\right) \frac{\left|\hat{\mathscr{C}}_i\right|}{\left|\hat{\mathscr{C}}\right|}$$

8.2. Experiments and data sets

Although ME and All−Three algorithms are provably correct, but verifying the accuracy of an implementation is a challenging task. With no prepossessing steps, all these algorithms are implemented in C++. The experiments were carried out on several publicly available real data sets which cover a spectrum of different data characteristics and sizes. We have used four different data sets in our experiments. Three real data sets from various sources often used in related research, and the fourth large data set was generated synthetically (artificially) using the near-duplicate generator (NDG) algorithm as described below. NDG generates ten of millions of near-duplicates for each set of real data.

Algorithm 2Near-Duplicate Generator (NDG) Algorithm

1: Remove a random number of characters from the data set record
2: Replace a random number of characters with others
3: Flip the first and last attributes (i.e., flip the first and last names in the lists)
4: Duplicate a random character. This might be done to more than one character
5: Abbreviate randomly - Keep the first character but this might be duplicated or removed as above

We ran our experiments on four categories of data set as follows.

Voting[1] We used the list of registrar voters in British Columbia as our original small data set with no duplicates. The original total number of records in the voting list is 225 and 975 records. Then we complement each of the two original voting data sets with a set of near-duplicates generated by the NDG algorithm. Additionally, we augmented and topped the voting data set to 1977 and 3745 references, respectively.

Cora[2] Cora data set contains bibliographic records and citations in scientific papers classified in several classes. The cora citation data set, which consists of a data set of original references and research papers, is often used in the duplicate detection community. Additionally, we augmented and topped the cora data set to 21,152 references and 32,005, a substantial larger data set for our experiments.

DBLP[3] DBLP is the bibliography database for computer science records from the DBLP web site. Each record is a concatenation of author names(s), title of the publication, some keywords, an abbreviated reference format citation (i.e., journal, book, editor). It consists of 43,935 real objects, both for relational and XML data. Additionally and for our experiments, we augmented the data set to 63,553 references.

Synthetic[4] The near-duplicate generator algorithm (NDG) is capable of algorithmically generating tens of millions of synthetic records from a set or original records. For instance, the voting, cora and DBLP data sets have been scaled up and topped by tens of thousands of records. The synthetic data set of 573,879 has been augmented and topped by NDG to an average of 352.992 records, for an average total of 926.871 records over three runs.

We generated a random number (0–8) of near-duplicates for each record using the NDG algorithm for the voting and DBLP data sets, and a random number (9–20) for the cora data set. Finally, all generated near-duplicates as explained above in the NDG algorithm are appended to the original data set file. Moreover, we ran the NDG algorithm three times, for each original real data, to generate three different sets of data (see Tables 1-6).

[1] http://ww.rootsweb.ancestry.com/canbc.vote898/votea.html.
[2] http://www.cs.umass.edu and mccallum/code-data.html.
[3] http://www.informatik.uni-trier.de/ley/db/.
[4] Algorithm 2: The near-duplicate generator (NDG) algorithm.

Table 1
Detection of near-duplicates in voting list1 data sets.

Voting lists	List1 Real Data Set 255			Average
Runs	run_1	run_2	run_3	Avg
NDG near-duplicates	1058	947	1002	1002
Real data & NDG near-duplicates	1313	1202	1257	1257
ME near-duplicates	1142	1003	1078	1074
CT near-duplicates	1054	987	1056	1032
VT near-duplicates	1098	988	1003	1029
FT near-duplicates	1064	968	998	1010

Table 2
Detection of near-duplicates in voting list2 data sets.

Voting lists	List2 Real Data Set 975			Average
Runs	run_1	run_2	run_3	Avg
NDG near-duplicates	1857	1901	1840	1866
Real data & NDG near-duplicates	2832	2876	2815	2841
ME near-duplicates	1633	1818	1702	1717
CT near-duplicates	1793	1877	1823	1831
VT near-duplicates	1859	1856	1803	1839
FT near-duplicates	1855	1902	1857	1871

Table 3
Detection of near-duplicates in Cora data sets.

Cora Data Set	Cora Real Data Set: 21,152			Average
Runs	run_1	run_2	run_3	Avg
NDG near-duplicates	9320	12,754	10,487	10,853
Real data & NDG near-duplicates	30,472	33,906	31,639	32,005
ME near-duplicates	11,412	11,674	10,078	11,054
CT near-duplicates	9234	12,657	10,456	10,782
VT near-duplicates	9341	12,788	10,501	10,876
FT near-duplicates	9289	12,768	10,485	10,847

Table 4
Detection of near-duplicates in DBLP data sets.

DBLP Data Set	DBLP Real Data Set: 43,935			Average
Runs	run_1	run_2	run_3	Avg
NDG near-duplicates	21,134	19,345	18,376	19,618
Real data & NDG near-duplicates	65,069	63,280	62,311	63,553
ME near-duplicates	22,160	20,018	19,102	20,426
CT near-duplicates	21,203	19,177	18,723	19,643
VT near-duplicates	21,119	19,256	18,303	19,559
FT near-duplicates	21,147	19,257	18,362	19,589

Table 5
Detection of near-duplicates in Synthetic data sets.

Synthetic Data Set	Synthetic Data Set: 573,879			Average
Runs	run_1	run_2	run_3	Avg
NDG near-duplicates	344,516	364,890	349,571	352.992
Real data & NDG near-duplicates	918,395	938,769	923,450	926.871
ME near-duplicates	348,602	361,768	352,459	354.276
CT near-duplicates	342,823	363,239	348,697	351.586
VT near-duplicates	343,945	363,885	348,880	352.236
FT near-duplicates	344,323	364,299	349,105	352.575

8.3. Voting lists data sets analysis

First, we consider a small set of records extracted from the voting list1 and list2 of 255 and 975 individuals, respectively. Then we ran three times the NDG algorithm on the voting lists where each run has generated a larger list of near-duplicates. For instance, run_1 generated 1058 near-

Table 6
ME, CT, VT and FT algorithms: Performance evaluation of purity, inverse purity and F-measure.

Data set	near-duplicates	Purity	Inverse	F-measure
	Performance		Purity	
Voting List1	ME	0.893	0.839	0.702
	CT	0.986	0.913	0.956
	VT	0.980	0.952	0.968
	FT	0.996	0.998	0.997
Voting List2	ME	0.812	0.907	0.821
	CT	0.965	0.988	0.929
	VT	0.989	0.995	0.967
	FT	0.996	0.899	0.998
Cora	ME	0.925	0.97	0.890
	CT	0.968	0.988	0.929
	VT	0.986	0.992	0.943
	FT	0.978	0.993	0.998
DBLP	ME	0.815	0.934	0.867
	CT	0.978	0.988	0.932
	VT	0.996	0.985	0.985
	FT	0.999	0.979	0.998
Synthetic	ME	0.945	0.957	0.921
	CT	0.937	0.986	0.959
	VT	0.964	0.983	0.994
	FT	0.982	0.990	0.991

duplicates for a total of 1313 records. That is, the original real voting list1 has been topped by 1058 near-duplicates (Table 1). Similarly, the original real voting list2 has been topped by 1857 for a total of 2832 (Table 2).

Both tables show the results of the NDG algorithm on the voting lists over three runs. We carried out the experiments by running the four algorithms, ME, CT, VT, and FT on the total number of records. The real data is topped by duplicates generated by NDG (4th row of Tables 1 and 2). Then we checked whether these algorithms identify the near-duplicates accurately. ME and All−Three algorithms performed more or less accurately on small data set sizes (1257 and 2841 voting records) because of the small amount of noise added to the data set. For instance, on small-sized data such as list2, CT and VT algorithms show an accuracy of 98.34% on the average with 31 false positives ((1831/1866 + 1839/1866)/2 = 98.34%). However, FT algorithm's accuracy is 100.27% where the extra 0.27% represents few false negatives on the average in list2 (Figs. 1 and 2 in Appendix). The ME algorithm accuracy is 92.02% with 149 false positives. Still, the ME accuracy is behind the performance of All−Three Algorithms.

8.4. Cora and DBLP data sets analysis

On the other types, medium- and large-sized data, All−Three algorithms consistently outperformed the ME algorithm in terms of accuracy. On the negative side, the ME algorithm added 2092 false negatives in run_1, but converged to 201 false negatives on the average due to readjusting the threshold to 0.8 set by the algorithm (Fig. 3 in Appendix). Overall, the ME algorithm added an extra 1.85% of false negatives over the 32,005 records (Table 3). The Merge−Filter−RC paradigm complemented by the comparison and construction of cluster representatives are very noticeable with a high similarity threshold which helped to improve the accuracy of All−Three algorithms (Table 3, Fig. 3 in Appendix). This has been shown in the FT algorithm with only an average of 6 false positives (99.95% of accuracy) on the cora data set of 32,005 records. Furthermore, the FT algorithm added only 29 false positives (99.85 of accuracy) on the DBLP data set of 63,553 records (Table 4, Figure 4 in Appendix). The performance of All−Three algorithms is the most substantial on DBLP with an average of 63,553 records where merging and filtering clusters' representatives is accurately achieved throughout the similarity variable and function thresholds. In particular, the VT and FT algorithms falsely detected an average of only 59 and 29 false positives, respectively. That is, an accuracy of 99.70% and 99.86% (Figs. 3 and 4 in Appendix). However, the ME algorithm is far behind with an average of 808 (4.12%) false negatives.

8.5. Synthetic data set analysis

Another important observation is that the tuning parameters that we inserted in both ME and SW algorithms overall added some extra significant accuracy to the near-duplicate detection. For instance, and over the 926,871 synthetic data set (Table 5, Figure 5 in Appendix). All−Three algorithms detected only 860 (0.092%) false negatives. That is, an accuracy of 99.90%. The ME algorithm performed also quite well over 926,871 synthetic large-sized data set with 1284 (0.14%) false negative, that is, an accuracy of 99.86%. Overall, ME and CT algorithms had missed to identify a very small number of false positives. That is, 0.14% and 0.15% over 926,871 records. On the DBLP data set, All−Three algorithms, in particular VT and FT algorithms, outperform the ME algorithm.

8.6. Performance evalution: purity, inverse purity and F-measure

With an extensive experimental analysis, the purity, inverse and F-measure achieved a value of nearly 1.0, a precision which outperforms the seminal work of Monge-Elkan. For instance, in the ME algorithm which is based on Jaro-Winkler's metric and using the attribute name provided by DBLP, the maximum F-measure is at threshold 0.8 and robust up to 0.9. The precision drops steadily below 0.8 due to many false positives. However, with the same algorithm based on Smith-Waterman's metric, the maximum F-measure is at threshold 0.9 and the SW precision is within the interval [0.9⋯1.0] due to prefixes and suffixes which are ignored in some references. For the same algorithm and for different metrics (i.e., Jaro-Winkler, Smith-Waterman, Levenshtein), the performance in terms of F-measure, precision and recall are quite different if we consider only the attribute affiliation in the experiment (Table 6, Figure 6 thru Figure 10 in Appendix). As needed, the near optimal threshold was not evaluated only once, but reevaluated and reconfigured if necessary as records are added to clusters as illustrated in the set of figures in appendix.

9. Conclusion

In this paper, we have introduced a near-duplication detection framework by improving Monge-Elkan's algorithm and also including an affine variant of the Smith-Waterman's algorithm to reduce the number of record comparisons. We have investigated and implemented a family of algorithms – constant threshold (CT), variable threshold (VT) and function threshold (FT) which are collectively referred to as All−Three algorithms, all based on the merge-filter cluster representatives Merge−Filter−RC approach. Our experiments with real-world and generated data sets have shown substantial gains in accuracy and potential efficiency to detect near-duplicates in very large data sets and across different domains.

As we have predicted in this investigation, the performance of Monge-Elkan's algorithm which is mainly evaluated by the number of near-duplicates within their correct clusters would not be as good and accurate as expected. This is due to several flaws in the algorithm, including priority queue and union set structures, record comparison algorithms, restricted clustering methods, and other parameters. The Merge−Filter−RC approach, complemented with the three classes of algorithms achieves a value of nearly 1.0, a precision which is nearly perfect and outperforms the seminal work of Monge-Elkan. In each run of the experiment, the major evaluation metric is the accuracy measured in terms of near-duplicate clusters. The effect of imperfection in terms of number of clusters and accuracy is reflected in cluster miss-classification with increasing data set sizes in Monge-Elkan's algorithm. All of these affect the precision, recall, and F-measure metrics. We believe that the cutoff and optimization values of threshold with other key tuning

parameters throughout semi-supervised machine learning will significantly improve the quality of the accuracy and effectiveness of detecting near-duplicate clusters. This is being investigated as the likely direction and outlook of our current research.

Acronyms

Merge−Filter−RC	Merge-Filter Representative-based Clustering
CT	Constant Threshold
VT	Variable Threshold
FT	Function Threshold
All−Three	Constant Threshold, Variable Threshold, Function Threshold
ME	Monge-Elkan
SW	Smith-Waterman
SNM	Sorted Neighborhood Method
NDG	Near-duplicate Generator
$\mathrm{OPT}(\mathcal{A})$	Optimal Local Alignment $(\mathcal{A})$
$\mathrm{NEAR}-\mathrm{OPT}_{\mathrm{DUPTREE}}(\hat{\mathscr{C}},i)$	near-optimal duplicate subtree for the node $\hat{\mathscr{C}}$ using i clusters

Credit author statement

CRediT (Contributor Roles Taxonomy) was introduced with the intention of recognizing individual author contributions, reducing authorship disputes and facilitating collaboration. The idea came about following a 2012 collaborative workshop led by Harvard University and the Wellcome Trust, with input from researchers, the International Committee of Medical Journal Editors (ICMJE) and publishers, including Elsevier, represented by Cell Press.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

I acknowledge the collaboration of Dr. Maamir Allaoua who collaborated with me on several interrelated published papers. He is now retired from the Dept. of Computer Science, University of Sharjah, Sharjah UAE. I would also like to thank two graduate students, Sepideh Pashami and Serveh Ghaderi, who spent one entire term carrying out the experiments and programming parts reported in this paper. I also thank Professors A. Elmagarmid and V. Verykios for providing me with some of the original benchmark data sets.

Appendix

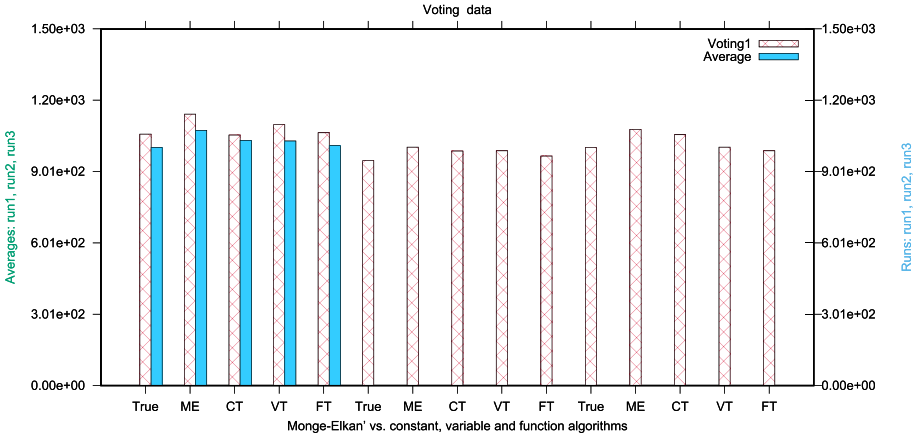


Fig. 1. Voting list1 data set: True duplicates vs. ME, CT, VT, and FT Algorithms

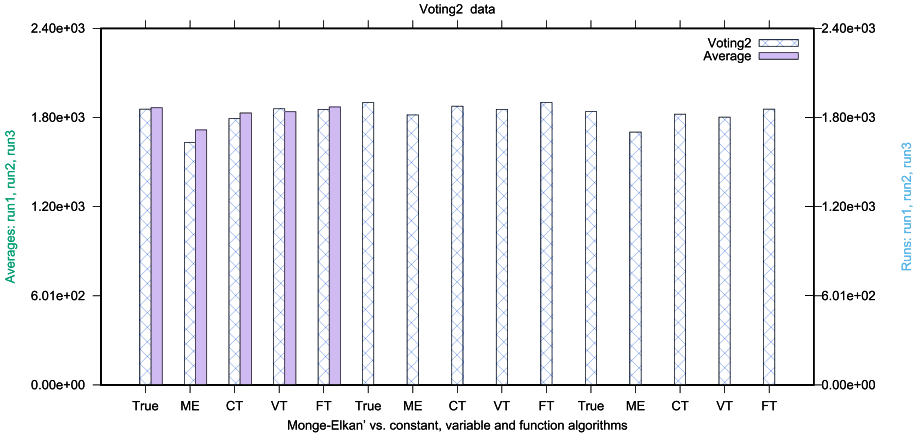


Fig. 2. Voting list2 data set: True duplicates vs. ME, CT, VT, and FT Algorithms

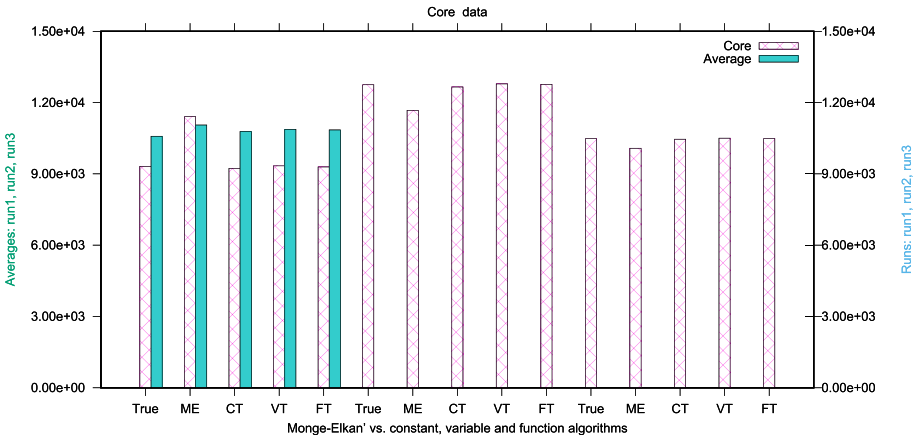


Fig. 3. Cora data set: True duplicates vs. ME, CT, VT, and FT Algorithms.

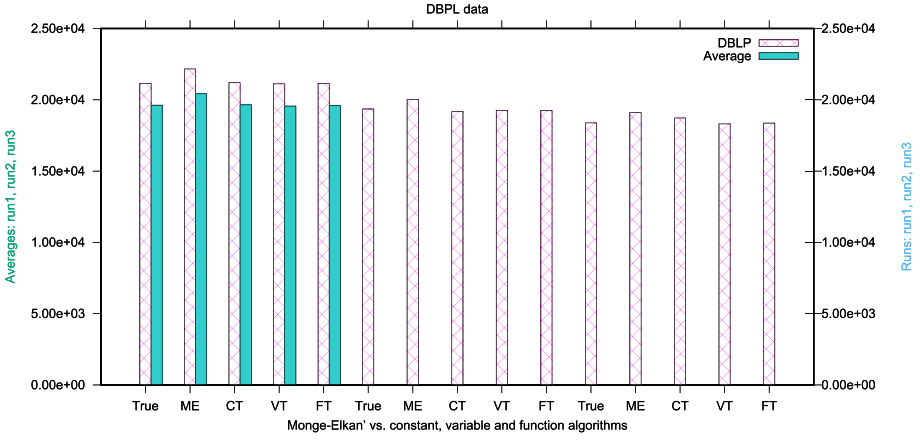


Fig. 4. DBLP data set: True duplicates vs. ME, CT, VT, and FT Algorithms.

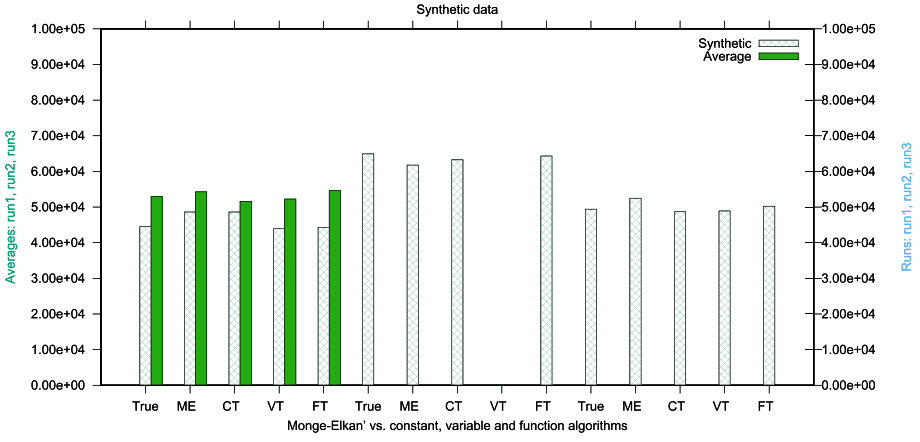


Fig. 5. Synthetic data set: Optimal duplicates vs. ME, CT, VT, and FT Algorithms

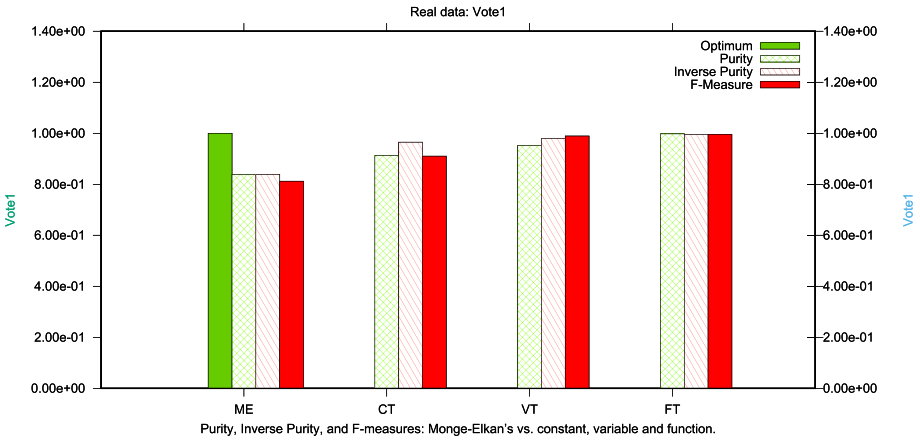


Fig. 6. Voting list1 data set: Optimal measures vs. ME, CT, VT, and FT measures

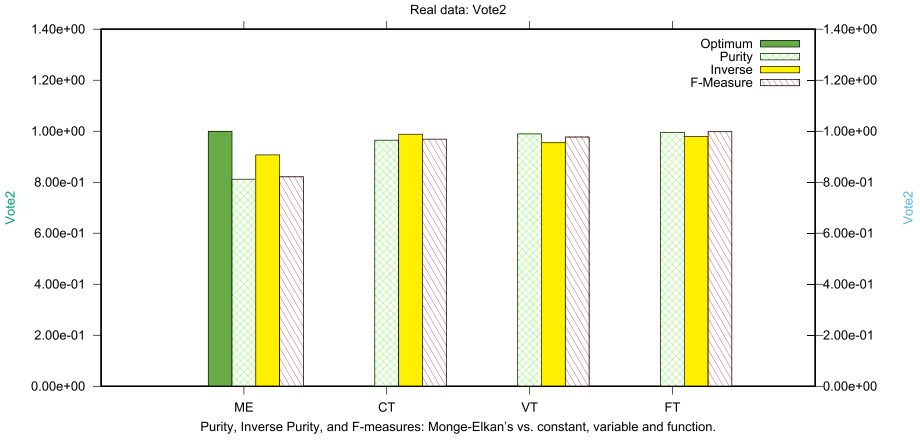


Fig. 7. Voting list2 data set: Optimal duplicates vs. ME, CT, VT, and FT measures

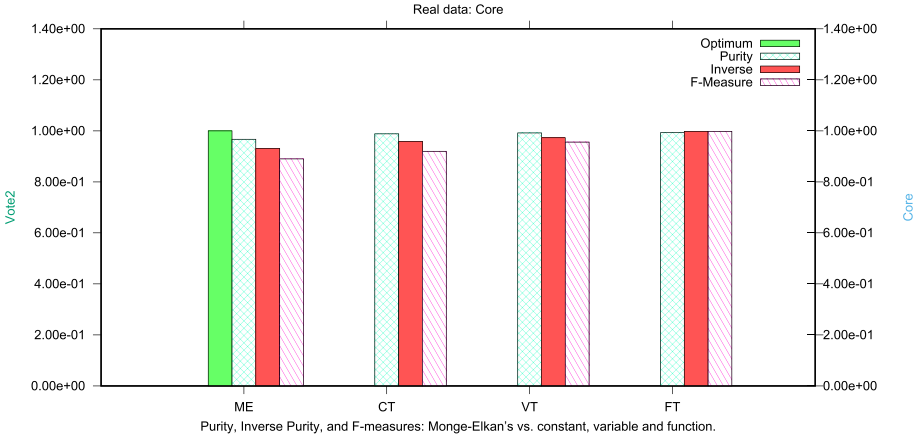


Fig. 8. Core data set: Optimal duplicates vs. ME, CT, VT, and FT measures

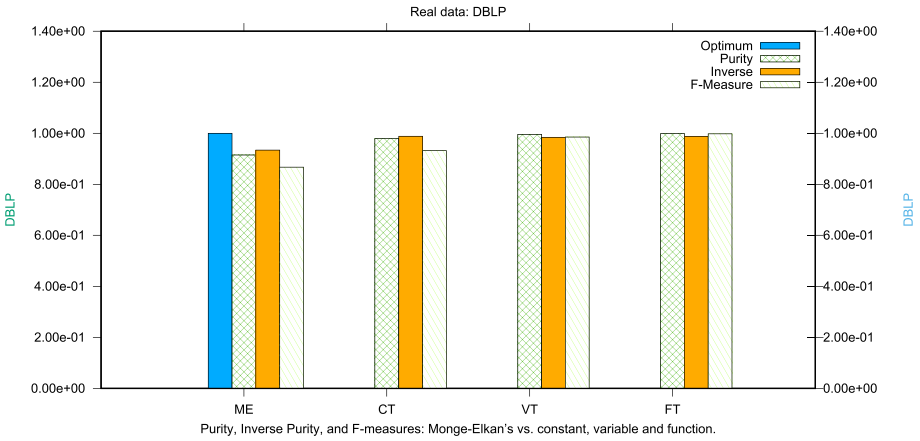


Fig. 9. DBLP data set: Optimal duplicates vs. ME, CT, VT, and FT measures

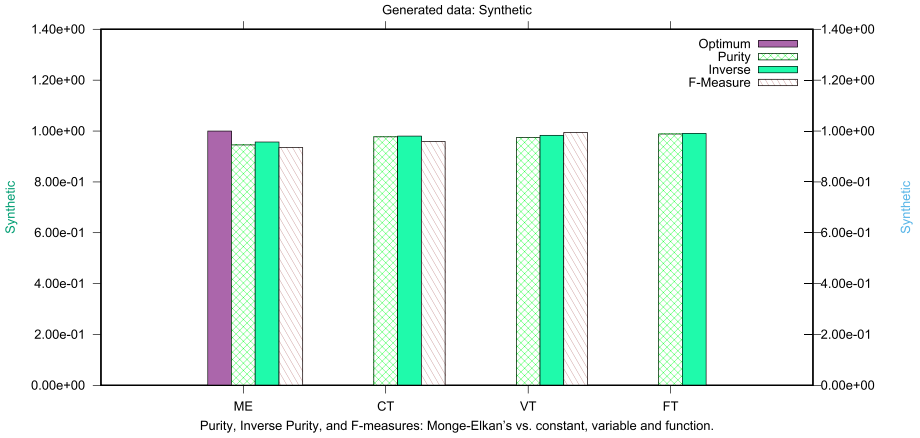


Fig. 10. Synthetic data set: Optimal duplicates vs. ME, CT, VT, and FT measures

References

[1] Newcombe H, Kennedy J, Axford S, James A. Automatic linkage of vital records. J Sci 1959;130(3881):954–9.
[2] Christen P. A survey of indexing techniques for scalable record linkage and deduplication. IEEE Transactions on Knowledge and Data Engineering (TKDE 2012; 24(9):1537–55.
[3] Jaro M. Advances in record-linkage methodology as applied to matching. J Am Stat Assoc 1989;84(406):414–20.
[4] Monge A. Adaptive detection of approximately duplicate database records and the database integration approach to information discovery. Ph.D. thesis, Ph.D. Thesis. San Diego: Dept. of Comp. Sci. and Eng., Univ. of California; 1997.
[5] Whang S, Menestrina D, Koutrika G, Theobald M, Garcia-Molina H. Entity resolution with iterative blocking. In: Proceedings of the ACM international conference on management of data. SIGMOD); 2009. p. 219–32.
[6] Weis M, Naumann F, Brosy F. A duplicate detection benchmark for xml (and relational) data. In: Proceedings of the SIGMOD inter. Workshop on information quality for information systems. IQIS); 2004. p. 10–9.
[7] Yan S, Lee D, Kan M, Giles L. Adaptive sorted neighborhood methods for efficient record linkage. In: Proceedings of the 7th ACM/IEEE-CS joint conf. on Digital libraries; 2007. p. 185–94.
[8] Hernandez M, Stolfo S. The merge/purge problem for large databases. In: Proceedings of the ACM SIGMOD international conference on management of data; 1995. p. 127–38.
[9] Papenbrock T, Naumann F, Heise A. Progressive duplicate detection. IEEE Trans Knowl Data Eng 2018;27(5). 1316–132.
[10] Draisbach U, Naumann F. On choosing thresholds for duplicate detection. In: Proceedings of the 18th international conference on information quality. ICIQ); 2013. p. 37–45.
[11] Elmagarmid A, Ipeirotis P, Verykios V. Duplicate record detection: a survey. IEEE Trans Knowl Data Eng 2007;19(1):1–16.
[12] Chen Q, Zobel J, Verspoor K. Duplicates, redundancies and inconsistencies in the primary nucleotide databases: a descriptive study, Jounal of Biological databases and curation. 2017. p. 2–16. https://doi.org/10.1093/database/baw163.

[13] Xiao C, Wang W, Lin X, Yu J, Wang G. Efficient similarity joins for near-duplicate detection. ACM Trans Database Syst 2011;36(3):15–41.
[14] Monge A, Elkan C. Domain-independent algorithm for detecting approximately duplicate database records. In: Proceedings of the ACM SIGMOD workshop on research issues on data mining and knowledge discovery. DMKD); 1997. p. 23–9.
[15] Fellah A, Maamir A. A domain independent methodology for near-duplicate detection. In: Proceedings of the international conference on applied computing, madrid Spain; 2012. p. 139–46.
[16] Navarra G. A guided tour to approximate string matching. ACM Comput Surv 2001; 33(1):31–88.
[17] D. Moreira, al, Image provenance analysis at scale, IEEE Trans Image Process 27 (12).
[18] Fellah A, Maamir A. Near-optimal domain independent approach for detecting duplicates. In: Proceedings of the 19th international conference on data mining, multimedia and image processing. Paris France; 2017. p. 2633–42.
[19] Bharambe D, Jain S, Jain A. A survey: detection of duplicate record. International Journal of Emerging Technology and Advanced Engineering 2012;2(11):298–307.
[20] Hassanian-esfahani R, Kargar Mj. Sectional minhash for near-duplicate detection. Expert Syst Appl 2018;99(1):203–12.
[21] Herschel M, Naumann F, Szott S, Taubert M. Scalable iterative graph duplicate detection. IEEE Trans Knowl Data Eng 2012;4(11). 2294–2108.
[22] Naumann F, Herschel M. An introduction to duplicate detection. Synthesis Lectures on Data Management 2010;2(1):1–87.
[23] Winkler W. Overview of record linkage and current research directions. A Statistical Research Division, U.S. Census Bureau; 2006. p. 1–44.
[24] Winkler W. Approximate string comparator search strategies for very large administrative lists, A Statistical Research Report Series (Statistics 2005-02). U.S. Census Bureau; 2005. p. 1–9. 02.
[25] Baxter R, Christen P, Churches T. A comparison of fast blocking methods for record linkage. In: Proceedings of the ACM SIGKDD workshop on data Cleaning,Record linkage, and object consolidation; 2003. p. 25–8.
[26] Monge A. Matching algorithms within a duplicate detection system. IEEE Data Engineering Bulletin 2000;23(4):14–20.
[27] Yandrapally R, Stocco A, Mesbah A. Near-duplicate detection in web app model inference. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering; 2020. p. 186–97.
[28] Thyagharajan KK, Kalaiarasi G. A review on near-duplicate detection of images using computer vision techniques. Arch Comput Methods Eng 2021;28:897–916.
[29] Draisbach U, Naumann F, Szott S, Wonneberg O. Adaptive windows for duplicate detection. In: Proceedings of the IEEE 28th international conference on data engineering. ICDE); 2012. p. 1073–83.
[30] Jaro M. Probabilistic linkage of large public. Journal of Statistics in Medicine 1995; 84(406):414–20.
[31] Hernandez M, Stolfo S. Real-world data is dirty: data cleansing and the merge/ purge problem. Data Min Knowl Discov 1998;2(1):9–37.
[32] Kopcke H, Rahm E. Frameworks for entity matching: a comparison. Data Knowl Eng 2010;69(2):197–210.
[33] Papadakis G, Svirsky J, Gal A, Palpanas T. Comparative analysis of approximate blocking techniques for entity resolution. In: Proceedings of the VLDB endowment. PVLDB); 2016. p. 684–95.
[34] Papadakis G, Alexiou G, Papastefanatos G, Koutrika G. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. In: Proceedings of the VLDB endowment. PVLDB); 2015. p. 312–23.
[35] Vogel T, Heise A, Draisbach U, Lange D, Naumann F. Reach for gold: an annealing standard to evaluate duplicate selection results. ACM Journal of Data and Information Quality 2014;5:1–22.
[36] Vogel T, Naumann F. Instance-based "one-to-some" assignment of similarity measures to attributes. In: Proceedings of the international conference on cooperative information systems. CoopIS); 2011. 412–0420.
[37] Smith T, Waterman M. Identification of common molecular subsequences. J Mol Biol 1981;147. 195–107.
[38] Needleman S, Wunsch C. General method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 1970;48:443–53.
[39] Deepa K, Rangarajan R, Selvi M. Automatic threshold selection using pso for ga based duplicate record detection. Int J Comput Appl 2013;62(4):181–7.
[40] dos Santos J, Heuser A, Moreira V, Wives L. Automatic threshold estimation for data matching applications. Inf Sci 2011;181(13):2685–99.
[41] Cheng D, Kannan R, Vempala S, Wang G. A divide-and-merge methodology for clustering. ACM Trans Database Syst 2006;31(4):1499–525.
[42] Li M, Wang H, Li J, Gao H. Efficient duplicate record detection based on similarity estimation. In: Proceedings of the 11th inter. Conf. on Web-page and Information Management (WAIM); 2010. p. 595–607.