[Array 11 (2021) 100070](https://doi.org/10.1016/j.array.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

|  |  |
| --- | --- |
| A R T I C L E I N F O | A B S T R A C T |
| Keywords:  Near-duplicate detection  Near-duplicates  Approximate duplicates  Clustering  Data mining applications and discovery Data cleaning | 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 dynami- |

cally. 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 | 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

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 infor-mation retrieval, entity resolution [5], object identification [6] in ma-chine 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 seman-tically 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 er-rors, misspellings, missing data, and differences in abbreviations and

E-mail address: [afellah@nwmissouri.edu](mailto:afellah@nwmissouri.edu).

<https://doi.org/10.1016/j.array.2021.100070>

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 Univer-sity, 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 in-dividual, even though they are quite different if byte-by-byte compari-sons 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 ci-tations in a bibliography database such as DBLP, CompuScience, and

Received 20 February 2021; Received in revised form 5 May 2021; Accepted 12 May 2021   
Available online 27 May 2021   
2590-0056/© 2021 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

A. Fellah Array 11 (2021) 100070

CiteSeer. Several algorithms, in particular domain-dependent, for quan-tifying 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 deduplica-tion, and image database. A substantial body of research has been con-ducted 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� 1019comparisons. 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’ char-acteristics (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 suc-cessive 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 re-cords. The window uses the blocking scheme by including all records with similar keys (lexically nearby keys) in the same window. All records

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 be-tween 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 stan-dard 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 at-tributes 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 interac-tion 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 depen-dent 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 sub-stitution'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 tech-niques (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].

within each sliding window are considered as potential near-duplicates 3. Contributions

and then compared with each other, however, the window size is diffi-

cult 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 pro-

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| posed as an umbrella for SNM where blocking has been empirically | framework | as | Merge-Filter | Representative-based | Clustering | (Mer- |

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 identi-fication 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 investi-gated 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

2

A. Fellah Array 11 (2021) 100070

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 inter-vention. 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 represen-tatives rather than with all records in the cluster. Thus, records do not have to be compared to all other records, but to only cluster represen-tatives which are considered for subsequent comparisons. We introduce cluster representatives which retain the most relevant syntactic and se-mantics 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 dupli-cate 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(n2) to O(nm), where m and and n are the number of representatives and records, respectively and m ≪ 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 al-comparisons. We ran an extensive experimental study using several real gorithms are implemented in Cþþ and use the same data set to make fair 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 scienti-

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 mis-matches 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 ½0; 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 is-sues. 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 cost(gap) ¼ s cost of extending a gap, and l is the length of a gap in the alignment of two þ e � l, where s is the affine cost of starting a gap in an alignment, e is the strings. Usually affine gap penalizes gap extension less than gap opening (e < s) thus we decrease the penalty for contiguous mismatched sub-strings by using a single long gap over many short gaps. Since the dif-ferences 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 per-forms well to detect similarities when records have minor syntactical differences, including typographical errors, abbreviations, and trunca-tions. In fact, the Monge-Elkan's algorithm approximates the solution to the optimal assignment problem in combinatorial optimization. This

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| fic papers from multiple sources, typically an alphanumeric domain class. | approximation | is | a | reasonable | trade-off | between | accuracy | and |
| The parameters used in ME are mapped to such a class of applications | complexity. |

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

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:

|  |  |  |  |
| --- | --- | --- | --- |
| SW. | MongeElkanðx; yÞ ¼1 jxj | X | max sim j¼1;jyjðx½i�; y½i�Þ |
| The rest of the paper is organized as follows. Section 3 reviews and |
| summarizes the effectiveness of Monge-Elkan and Smith-Waterman al- |

gorithms. 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 pro-vides 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 con-ditions 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.

3

A. Fellah Array 11 (2021) 100070

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 U. Let n be the number of real-world entities over a plurality of large databases that consist of records Rn ¼ r1; r2; …; rn, where a large number of ri are potential near-duplicates. We denote by D ¼ D1; D2;…; Dd the set of problem domains. Similarity and distance measures are often normalized in the range ½0; 1�, and ½0; ∞� or ½0; 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: D1 � D2 → [0,1], such that sim(r1, r2) ¼ 0 if r1 and r2 are least similar and sim(r1, r2) ¼ 1 if r1 and r2 are identical. (D1 might be equal to D2).

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 neg-atives 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 ½0; 1�, is defined as 1 - (declared duplicates/all tuple pairs). The simi-larity threshold line in Fig. 1 ensures a trade-off among recall, precision and reduction ratio. If we shift the similarity threshold line (similarity

|  |  |  |
| --- | --- | --- |
| Definition 5.2. | A distance measure is a non-negative function dist: | threshold 1) to the right, consequently more tuple pairs will be rejected |

D1 � D2 → [0, 1], such that dist(r1, r2) ¼ 0 if r1 and r2 are exactly similar or identical, and dist(r1, r2) ¼ 1 if r1 and r2 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 re-cords 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

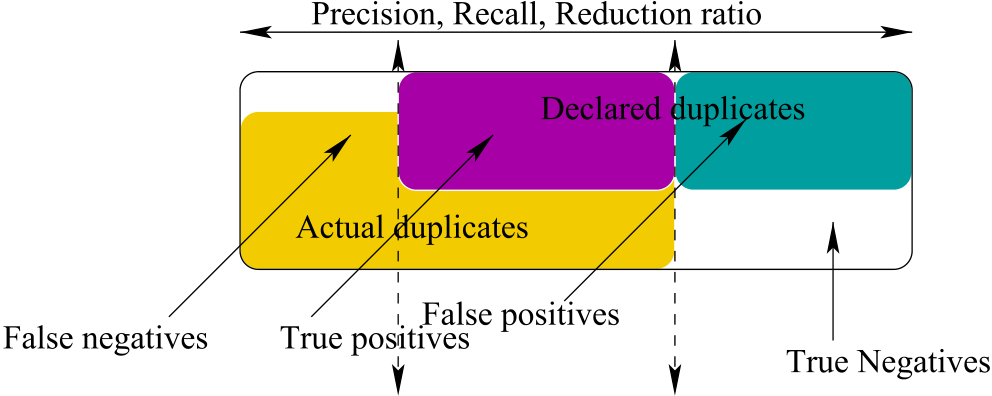




Fig. 1. Similarity threshold factors and tradeoff between prominent mea-sure metrics.

4

A. Fellah Array 11 (2021) 100070

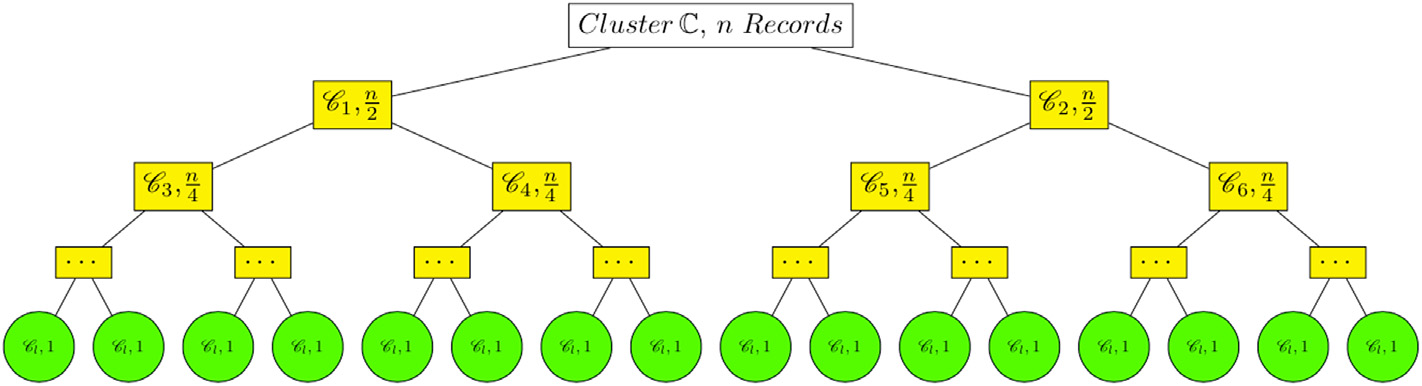


Fig. 2. Construction of a top-down cluster tree T.

of n records.

divide phase. Each internal node of T is a subset of the data set records. Let ℂ ¼ C 1; C 2; …; C n be the set of clusters of T produced by the

The left and right children of a node C i forms a partition of the parent such that∣ C i∣¼ n=2dand 1 � i, i � n (Fig. 2). Let T be a top-down cluster tree then for any two clusters we have either C i ⊂ C j or C j ⊂ C i or C i \ C j Starting at the leaves of T, the bottom-up merging phase is applied ¼ ∅, i 6¼ j, 1 � i, j � n.

recursively to each node of T towards the root enabling the construction

of a new tree of clusters whose root is^T as depicted in Fig. 3. To

accomplish this, the near-optimal cluster of an interior node^C i in the tree^T is obtained by merging and distilling dynamically the near-optimal

clusters of the left and right children of^C i. The result of clustering T is a partition^C ¼ ^C 1; ^C 2; …; ^C q where one or more ^C i are the nodes of ^T, referred to as near-optimal duplicate tree, i � q and q ≪ 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^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^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 (^C i, ^C j), and their

similarity score. Let such a set of cluster representatives denoted by^Ri ¼ f^ri1;^ri2;…;^rijg, 1 � j � l where l indicates the number of representatives

generated ith cluster^C i, represented as a node in^T, has a set of near-��� ^C i��� ¼ k and��� ^Ri��� ¼ l, where l is much less than k. Every for a fixed i.

duplicates and cluster representatives referred to as frijgk j¼1and f^rijgl j¼1, 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 rep-resentative(s) might be subsequently updated, either by retaining the same ones or iteratively re-computing and re-assigning new representa-tives. In addition, Merge�Filter�RC enforces transitivity between re-cords in a cluster ^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(n2) to O(nm), where m and and n are the number of representatives and records, respectively. m≪ n and m is always independent of n, but dependent on the class of the algorithm (constant, variable, or function).

of the appropriate objective function g and its parameters. We are The cornerstone property of Merge�Filter�RC is based on the choice

interested in finding locally the optimal clustering at each level of^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, OPTðAÞ, and quantifies the similarity in terms of a high-scoring alignment 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 ¼ x1…xp, y ¼ y1…yq where x, y 2 Σ\*. The state space of all alignments of x and y is the mapping g: ð〈x; y〉Þ 7! Z, the set of integers. The optimal local alignment score of the subsequences x1…xp and y1…yq is obtained by maximizing g among all alignments. That is,

|  |  |  |
| --- | --- | --- |
| respective cluster representatives (^Ri,^Rj) where 1 � i, j � n, then returns whether (^C i,^C j) are very similar, similar or not similar by finding the | g0ð〈x; y〉Þ ¼ max A | gð〈x; y〉Þ |

optimal local alignment using the SW algorithm with the maximum

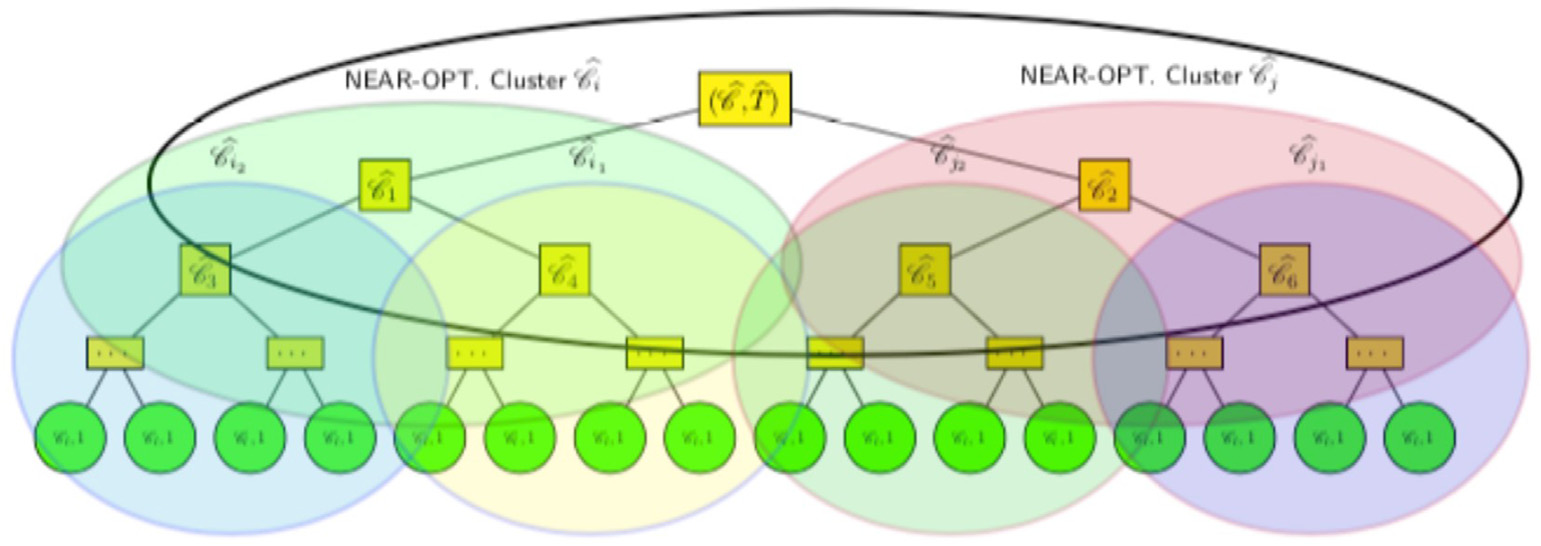


Fig. 3. Vizualization of a near-optimal duplicate cluster tree^T.

5

A. Fellah Array 11 (2021) 100070

OPTðAÞ ¼ argmax A gð〈x; y〉Þ

Let.

^C l and ^C r be the left and right children of an internal node ^C in ^T. Let NEAR � OPTDUPTREEð^C ; iÞ be the near-optimal duplicate sub-tree for the node^C using i clusters as stated recurrently in the following theo-rem. 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, DUPTREEð^C ; iÞ, facilitate the re-optimization (i.e., tuning) in the object function g to satisfy the proper-ties and accomplish the near-optimality solution. Thus, each subtree of

the node^C using i clusters, DUPTREEð^C ;iÞ, 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 corre-sponding objective function g. We explore the clustering methodology of [41] further and formally introduce near-optimal duplication tree of a

node objective function values of each alignment obtained by the SW algo- ^C using i cluster defined as NEAR � OPTDUPTREEð^C ; iÞ. The

rithm guarantee the optimal local alignment across all subtrees of^T.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Theorem 6.1. | NEAR � OPTDUPTREEð^C ;iÞ ¼ | | | �^C r;i�j� | if i ¼ 1 if i > 1 |
| � ^C NEAR�OPTDUPTREE | | �^C l;j� | [NEAR�OPTDUPTREE |

cluster. However, (r1, r3) and (r2, r3) would be non-duplicates. Now, consider a new record r4 added to R3 with sim(r1, r4) ¼ 0.75, sim(r2, r4) ¼ 0.88, and sim(r3, r4) ¼ 0.96. Then, (r2, r4) and (r1, r4) 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, (r1, r4) and (r2, r3), 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 accu-rately 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, NEAR � OPTDUPTREEð^TÞ gives the almost optimal clus-tering. We consider two upper and lower bound threshold values, θl and θu, 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 re-cord alignments. Thus, we set the semantic similarity threshold param-eter for which two records are considered semantically similar to θu. In the same way as semantically similar records, we set the non-similarity

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| where | �^C l;j� | [NEAR�OPTDUPTREE | �^C r;i�j�� | threshold to θl. |
| We set θu to 0.7 and θl to 0.5 for declaring two records as near- |
| j¼argmin 1�j<ig�NEAR�OPTDUPTREE |
| duplicate and non-duplicate, respectively. θ � θl produces loose simi-larity (not similar), θ �θu produces high similarity (very similar), and if θ falls between these values it is regarded as a regular similarity (similar). |

Proof. We proceed by induction on i. The base case handles all clusters of^T. That is, all initial clusters^C 1;^C 2; …;^C q with a single represen-tative 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

Let^C l and^C r be the left and right children of an internal node^C whose cluster representative is^R. We denote by^Rl and^Rr the left and right cluster representative children of^R. As a consequence of Theorem 6.1, we state the following result.

|  |  |  |
| --- | --- | --- |
| in the same cluster then they are considered to be near duplicate or | Corollary 6.1. | NEAR � OPTDUPTREEð^R;iÞ ¼ |
| 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(^T) be an internal node of^T, and^T1 and^T2 the left and

right subtrees recursively build from the leaf clusters of^T, respectively.

Without loss of generality, define^C 1 l and ^C 1 r to be the clusters whose

root is^T1; and^C 2 l and ^C 2 r be the clusters rooted in the right subtree ^T2,

ordered as ð^C 1 l; ^C 1 rÞ;ð ^C 2 l; ^C 2 rÞ. Denote by ( ^R1 l; ^R1 r) and ( ^R2 l; ^R2 r),

the set of cluster representatives of (^C 1 l; ^C 1 r) and ( ^C 2 l; ^C 2 r), respec-

tively. Let^R1

and

representative which is used for subsequent comparisons. Importantly,

representative clustering reduces the total number of record comparisons��� ^Ri r��� ¼ q > 1, for i ¼ 1, 2. That is, a cluster has more than one l ¼ ^R1 l [ ^R1 r and ^R2 r ¼ ^R2 l [ ^R2 r such that��� ^Ri l��� ¼ p > 1

substantially and furthermore representatives are bounded by two sim-

ilarity threshold values, θu and θl. Keeping this potential of multiple

cluster representatives, ^C l ¼ ^C 1 l [ ^C 1 r and ^C r ¼ ^C 2 l[ ^C 2 r. For

convenience, let^R1 l ¼ f^rl 1;1;^rl 1;2; …;^rl 1;pg and ^R2 l ¼ f^rl 2;1;^rl 2;2;…;^rl 2;pg.

Similarly^R1 are in the same cluster then they are considered to be near-duplicates or r and ^R2 r are defined. If two records (i.e., representatives)

exactly similar, and if not they are dissimilar. For instance, let R3 ¼ fr1;

r2; r3g be the set of records to be compared and the initial good threshold

chosen is in the interval ½0:83⋯0:90�. Assume we have the following

similarities: sim(r1, r2) ¼ 0.88, sim(r1, r3) ¼ 0.72, and sim(r2, r3) ¼ 0.80.

Then, (r1, r2) are classified as near-duplicates and assigned to the same

6

A. Fellah Array 11 (2021) 100070

^C q, where each ^C i is a node of ^T and q is unknown in advance. Without

loss of generality, let assume^C 1 l and ^C 1 r be the clusters whose root ^T1;

and^C 2 l and ^C 2 r be the clusters rooted in the right subtree ^T2 of ^T,

ordered as ð^C 1 l; ^C 1 rÞ;ð ^C 2 l; ^C 2 rÞ.

The procedure CONSTANT THRESHOLD (^C 1 l; ^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 al-

gorithm 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(n2)

to O(mn), where reduced m and n are the number of representatives and

records, respectively (m is much smaller than n).

Algorithm 1 (continued)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3: Compare (^C 1 | | | | l, ^C 2 | l) \(⊳\) comparison of ^C 1 | | | | l with ^C 2 | | l |
| 4: if (^C 1 | l, ^C 2 | | l) are very similar then | | | | | | | | |
| 5:^C new ← Merge (^C 1 | | | | | | l, ^C 2 | l) | | | | |
| l  6: Remove^C 2  7:^Rnew ←^R1 l \(⊳\) cluster rep. ← rep. of ^C 1  8: end if  9: if^C 1 l, ^C 2 l are similar then | | | | | | | | l | | | |
| 10:^Rnew ← Merge (^C 1 l l, ^C 2 l)  11: Remove^C 2  12:^Rnew ←^R1 l [ ^R2 l \(⊳\) cluster rep. ← rep. of ^C 1  13: end if | | | | | | | | | | l [ rep. of ^C 2 | |
| 14: if (^C 1 | | l, ^C 2 l | | l) are not similar then | | | | | | | |

l

15: no operation   
16: end if

|  |  |  |  |
| --- | --- | --- | --- |
| 7.2. Variable threshold (VT) algorithm | 17: Goto step 1 and compare (^C 1 | l, ^C 2 | r) |

|  |  |  |
| --- | --- | --- |
| In procedure VARIABLE THRESHOLD (^C 1 | l; ^C 2 | l) in algorithm 1, each |

cluster has only one representative and only one variable threshold value is considered, starting at θ1 ¼ 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

^C 1 l with ^C 2 l and ^C 2 r, then we compare ^C 1 r with ^C 2 l and ^C 2 r. Without loss of generality and for algorithmic simplicity, we assume

cluster^C 1 has ^r1 as a record representative and θ1 as a threshold. Sim-ilarity,^C 2 has ^r2 as a record representative and θ2 as a threshold. At start, clusters have one record and the threshold value θ1 is set to 0.5.

|  |  |  |  |
| --- | --- | --- | --- |
| 18: Goto step 1 and compare (^C 1 | | r, ^C 2 | l) |
| 19: Goto step 1 and compare (^C 1 | | r, ^C 2 | r) |
| 20: return (^R1 l; ^R2  21: end procedure | l)\(⊳\) returns cluster representatives | | |

1: procedure VARIABLE THRESHOLD (^C 1,^C 2) \(⊳\) VT algorithm   
2: Compare and find SW of ^r1 and ^r2 \(⊳\) at start θ1, θ2 are 0.5 for^C 1,^C 2 3: if (SW > θ1 or (SW > θ2) then   
4: if (θ1 > θ2) then   
5: Merge (^C 1,^C 2)   
6: Remove^C 2   
7: Set representative ← ^r1 \(⊳\) update representative to ^r1   
8: θ1 ← ðθ1 þSWÞ=2 \(⊳\) update θ1 9: end if

7.3. Function threshold (FT) algorithm 10: if (θ2 �θ1) then

11: Merge (^C 1,^C 2)

In this third category of algorithms each cluster has more than one

representative. Furthermore, two variable threshold values are consid-

ered, an upper bound value θu ¼ 0.7, and a calculated threshold value θc. 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^C 1 l with ^C 2 l and ^C 2 r,

then we compare^C 1 r with ^C 2 l and ^C 2 r. In addition, we compare ^C 1 r

with every cluster in the second half subtree.

max SW > θu The function COMPARE ð^C 1;^C 2Þ shows the very similar case when (lines 4 and 5). The other two cases, similar

(θl � max SW � θu) and not similar (max SW < θl), can also be treated

in a same manner. Lines 4 and 5 in the function COMPARE (^C 1;^C 2) should

be substituted by lines 4 and 5 in the function CHECK-SIMILARITY ð^C 1;^C 2Þ, 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 ð^C 1;^C 2Þ should be substituted by lines 7 and 8 in the function

CHECK-SIMILARITY ð^C 1;^C 2Þ. In the same way lines 11 and 12 should be substituted by lines 7 and

8. Let^R1 and^R2 be the set of the cluster representatives of^C 1 and^C 2,

rithmic convenience, let assume that q � p. Let^R1 ¼ f^r1igp��� ^R1��� ¼ p and��� ^R2��� ¼ q, where p, q � 1. For algo-respectively such that

f^r2jgq j¼1be the cluster representatives of ^C 1 and ^C 2, respectively. The

value max SW refers to the value returned by the Smith-Watermann

algorithm.

|  |
| --- |
| 12: Remove^C 1  13: Set representative ← ^r2 \(⊳\) update representative to ^r2 14: θ2 ← ðθ2 þ SWÞ=2 15: end if  16: else  17:^C 1 and^C 2 are not similar  18: no operation  19: end if  20: end procedure |

|  |
| --- |
| 1: procedure FUNCTION THRESHOLD (^C 1 l; ^C 2 l) \(⊳\) FT algorithm  2: Find max SW between^C 1 l and every cluster in ^T2  3: Suppose maxSW is between^C 1 l and ^C 2 r |

4: if (max SW > θu) then

|  |  |  |  |
| --- | --- | --- | --- |
| l 5: Merge (^C 1 | , ^C 2 | r) | r |
| 6: Remove^C 1 | l | |
| 7: Representative ← representative of^C 2 | | |

8: end if

|  |  |
| --- | --- |
| 9: if (θc � max SW < θu) then  10: Merge (^C 1 l, ^C 2 r) | |
| 11: Remove^C 2 | r |

12: Representative ← Union of representatives 13: end if   
14: if (max SW < COMPUTE(θc) then   
15: no operation   
16: end if   
17: end procedure

|  |
| --- |
| 1: function COMPARE (^C 1;^C 2) \(⊳\) very similar case 2: Compare and find SW of ^r11 with each f^r2jgq j¼1 3: Find max SW; m ← max SW |

4: if at any time (max SW > θu) then

|  |  |
| --- | --- |
| Algorithm 1 | 5: (^C 1,^C 2) are very similar  6: Stop and return max SW |

All � Three Algorithms.

1: Initialization: Two cluster leaf records (C l, C l)

|  |  |  |
| --- | --- | --- |
| 2: procedure CONSTANT THRESHOLD (^C 1 | l; ^C 2 | l) \(⊳\) CT algorithm |

7: end if   
8: if (max SW < θu) then   
9: Compare and find SW of ^r12 with each f^r2jgq j¼1   
10: Find maxSW; start with max SW equal to m   
11: if at any time (max SW > θu) then

(continued on next column) (continued on next page)

7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A. Fellah  Algorithm 1 (continued) | Array 11 (2021) 100070 | | | | | |
| j¼argmin 1�j<ig�NEAR�OPTDUPTREE | �^C | 2;j�l | [NEAR�OPTDUPTREE | �^C | 2;i�j��r |
| 12: (^C 1,^C 2) are very similar 13: Stop and return max SW 14: end if  15: end if |  |  |  |  |
| 8. Evaluation metrics and experiments | | | | | |

16: end function

|  |
| --- |
| 1: function COMPUTE (θc)  2: t ← 0.3 (1 �θu) \(⊳\) θu: Monge-Elkan's threshold  3: x ← the number of representative in^C 2 r � 1  4: Let m ← minimum(x, maximum number of allowed representatives) |

5: D ← t/(maximum number of allowed representatives) 6: θc ← 2 � t � m � D   
7: return θc   
8: end function

1: function CHECK SIMILARITY (^C 1;^C 2) \(⊳\) Check degree of similarity: \(⊳\) very similar, similar, not similar   
2: if (max SW > θu) then   
3: (^C 1,^C 2) are very similar   
4: end if   
5: if (max SW � θu) and (max SW � θl) then   
6: (^C 1,^C 2) are similar   
7: end if   
8: if (max SW < θl) then   
9: (^C 1,^C 2) are not similar   
10: end if   
11: end function

|  |
| --- |
| 1: function COMPARE LEFT-RIGHT SUBTREES (^C 1 l;ð ^C 2 l; ^C 2 rÞ)  2: Find max SW between^C 1 l and every cluster in ^T2 \(⊳\) similar to function COMPARE  3: Suppose max SW is between^C 1 l and ^C 2 r |

4: if (max SW > θu) then

|  |  |  |  |
| --- | --- | --- | --- |
| 5: Merge (^C 1 | l, ^C 2 | r) | r |
| 6: Remove^C 1 | |  |  | | --- | --- | | 1 | l | | |
| 7: Representative ← representative of^C 2 | | |

8: end if   
9: if (θc � max SW < θu) then   
10: Merge (^C 1 l, ^C 2 r)   
 r   
11: Remove^C 2   
12: Representative ← Union of representatives 13: end if   
14: if (max SW < θc) then   
15: do nothing   
16: end if   
17: end function

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Without loss of generality, let assume^C 1 l and ^C 1 r be the clusters  whose root^T1; and^C 2 l and ^C 2 r be the clusters rooted in the right  subtree^T2 of^T, ordered as ð^C 1 As a result of Theorem 6.1, we state the following results: l; ^C 1 rÞ;ð ^C 2 l; ^C 2 rÞ. | | | | | | | |
| Corollary 7.1. NEAR � OPTDUPTREEð^C 1 l;iÞ ¼  8 < : ^C 1  NEAR�OPTDUPTREE�^C  l  2;j�[NEAR�OPTDUPTREE l  where | | | | �^C | 2;i�j�r | | if i ¼ 1 if i > 1 |
| j¼argmin 1�j<ig�NEAR�OPTDUPTREE | �^C | 2;j�l | [NEAR�OPTDUPTREE | | | �^C | 2;i�j��r |
| Corollary 7.2. NEAR�OPTDUPTREEð^C 1 r; iÞ¼  8 < : ^C 1  NEAR�OPTDUPTREE�^C  l  2;j�[NEAR�OPTDUPTREE r  where | | | | �^C | 2;i�j�r | | if i ¼ 1 if i > 1 |

8

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A. Fellah | | | | | |  |  | | --- | --- | | ��� ^C | ��� | | Array 11 (2021) 100070 | | | | | |
| F � measure ¼ | q  X | F | �^C i; C j | � | Table 1 | | | | | |
| Detection of near-duplicates in voting list1 data sets. | | | | | |
| X |  |  | ^C | Voting lists | List1 Real Data Set 255 | | | Average | |
| 8.2. Experiments and data sets | | | | |  | Runs | run1 | run2 | | run3 | Avg |
|  | NDG near-duplicates | 1058 | | 947 | 1002 | 1002 |

verifying the accuracy of an implementation is a challenging task. With Although ME and All�Three algorithms are provably correct, but

no prepossessing steps, all these algorithms are implemented in Cþþ. 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-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| duplicate generator (NDG) algorithm as described below. NDG gener- | Voting lists | List2 Real Data Set 975 | | |  | Average |
| ates ten of millions of near-duplicates for each set of real data. | Runs | run1 | run2 | | run3 | Avg |
| Algorithm 2Near-Duplicate Generator (NDG) Algorithm | NDG near-duplicates | 1857 | | 1901 | 1840 | 1866 |
| Real data & NDG near-duplicates | 2832 | | 2876 | 2815 | 2841 |
| ME near-duplicates | 1633 | | 1818 | 1702 | 1717 |
| 1: Remove a random number of characters from the data set record | CT near-duplicates | 1793 | | 1877 | 1823 | 1831 |
| 2: Replace a random number of characters with others | VT near-duplicates | 1859 | | 1856 | 1803 | 1839 |
| 3: Flip the first and last attributes (i.e., flip the first and last names in the lists) | FT near-duplicates | 1855 | | 1902 | 1857 | 1871 |

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 | Table 3 |

Detection of near-duplicates in Cora data sets.

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

Voting1We 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 top-ped the voting data set to 1977 and 3745 references, respectively.

Cora2Cora 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.

DBLP3DBLP is the bibliography database for computer science re-cords from the DBLP web site. Each record is a concatenation of author names(s), title of the publication, some keywords, an abbreviated refer-ence format citation (i.e., journal, book, editor). It consists of 43,935 real objects, both for relational and XML data. Additionally and for our ex-periments, we augmented the data set to 63,553 references.

Synthetic4The 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cora Data Set | Cora Real Data Set: 21,152 | | | Average |
| Runs | run1 | run2 | run3 | 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 | run1 | run2 | run3 | 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.

|  |  |  |  |
| --- | --- | --- | --- |
| an average of 352.992 records, for an average total of 926.871 records | Synthetic Data Set | Synthetic Data Set: 573,879 | Average |

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.u](http://www.cs.umass.edu)[ni-trier.de/](http://www.informatik.uni-trier.de/)ley/db/.

4 [Algorithm 2: The near-duplicate gen](http://www.informatik.uni-trier.de/)erator (NDG) algorithm.

9

A. Fellah Array 11 (2021) 100070

Table 6 (Figs. 3 and 4 in Appendix). However, the ME algorithm is far behind

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 |
| Voting List2 | CT | 0.986 | 0.913 | 0.956 |
| VT | 0.980 | 0.952 | 0.968 |
| FT | 0.996 | 0.998 | 0.997 |
| ME | 0.812 | 0.907 | 0.821 |
| Cora | CT | 0.965 | 0.988 | 0.929 |
| VT | 0.989 | 0.995 | 0.967 |
| FT | 0.996 | 0.899 | 0.998 |
| ME | 0.925 | 0.97 | 0.890 |
| DBLP | CT | 0.968 | 0.988 | 0.929 |
| VT | 0.986 | 0.992 | 0.943 |
| FT | 0.978 | 0.993 | 0.998 |
| ME | 0.815 | 0.934 | 0.867 |
| CT | 0.978 | 0.988 | 0.932 |
| VT | 0.996 | 0.985 | 0.985 |

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 sig-nificant 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 ac-curacy 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 al-gorithms, in particular VT and FT algorithms, outperform the ME algorithm.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Synthetic | FT | 0.999 | 0.979 | 0.998 | 8.6. Performance evalution: purity, inverse purity and F-measure |
| 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 orig-inal 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 al-gorithms, 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 þ 1839/ 1866)/2 ¼ 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

rithms consistently outperformed the ME algorithm in terms of accuracy. On the other types, medium- and large-sized data, All�Three algo-

On the negative side, the ME algorithm added 2092 false negatives in run1, but converged to 201 false negatives on the average due to read-justing 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 com-plemented 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 Ap-pendix). 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%

10

A. Fellah Array 11 (2021) 100070

parameters throughout semi-supervised machine learning will signifi-cantly 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.

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.

Acronyms Declaration of competing interest

Merge�Filter�RC Merge-Filter Representative-based Clustering CT Constant Threshold

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence

VT Variable Threshold the work reported in this paper.

FT Function Threshold

All�Three Constant Threshold, Variable Threshold, Function Threshold ME Monge-Elkan Acknowledgments

SW Smith-Waterman I acknowledge the collaboration of Dr. Maamir Allaoua who collab-

SNM Sorted Neighborhood Method orated with me on several interrelated published papers. He is now

NDG Near-duplicate Generator retired from the Dept. of Computer Science, University of Sharjah,

OPTðAÞ Optimal Local Alignment ðAÞ

NEAR � OPTDUPTREEð^C ;iÞ near-optimal duplicate subtree for the node

^C using i clusters

Credit author statement

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.

CRediT (Contributor Roles Taxonomy) was introduced with the Appendix

intention of recognizing individual author contributions, reducing   
authorship disputes and facilitating collaboration. The idea came about

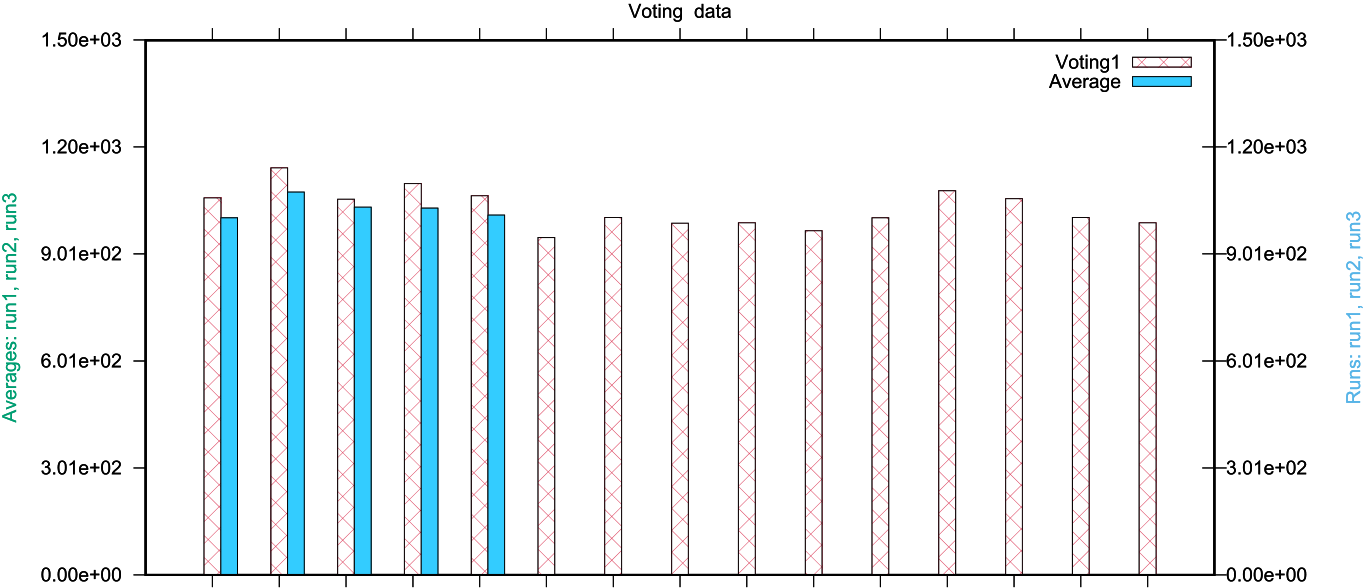






Fig. 1. Voting list1 data set: True duplicates vs. ME, CT, VT, and FT Algorithms

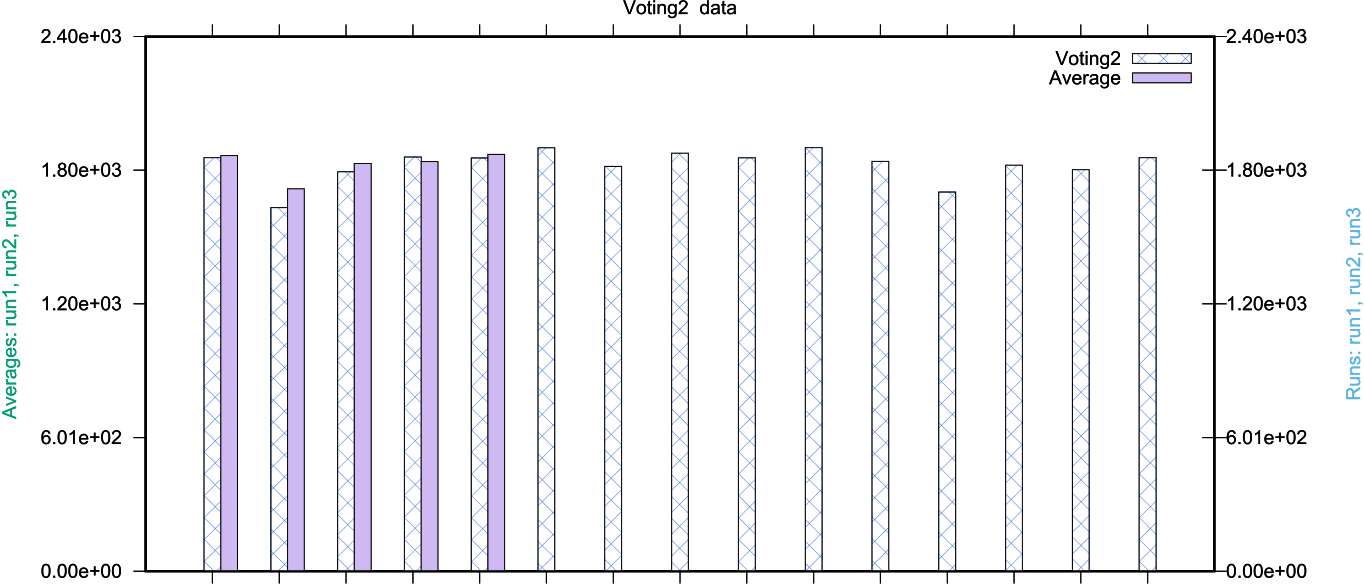






Fig. 2. Voting list2 data set: True duplicates vs. ME, CT, VT, and FT Algorithms

11

A. Fellah Array 11 (2021) 100070



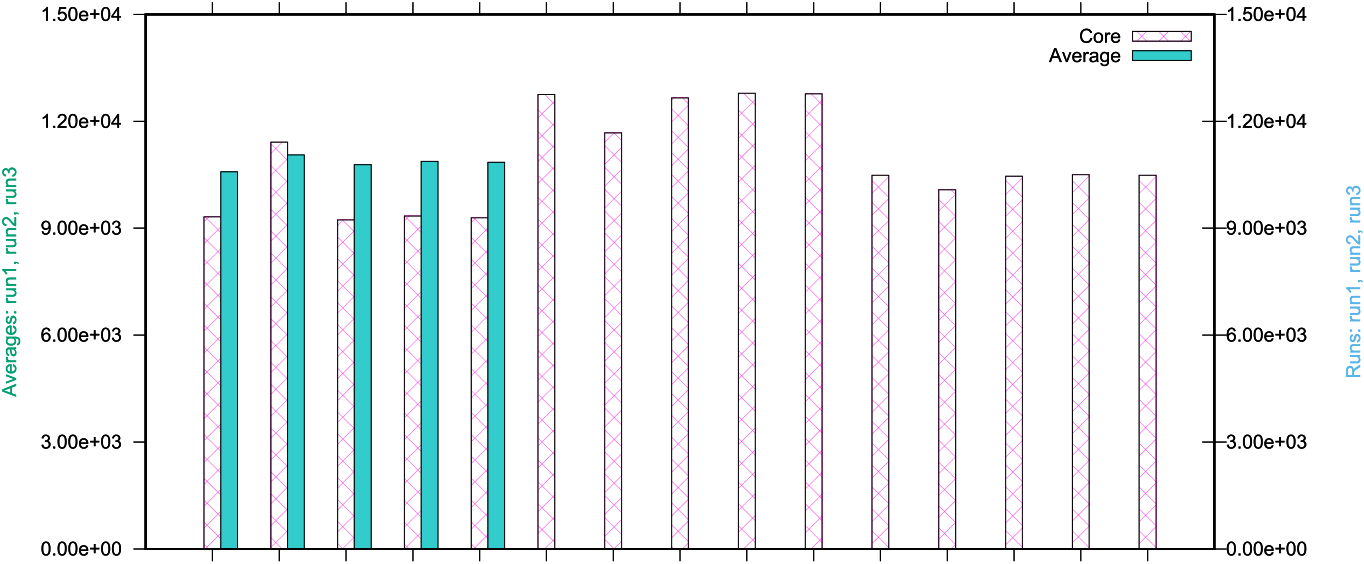






Fig. 3. Cora data set: True duplicates vs. ME, CT, VT, and FT Algorithms.

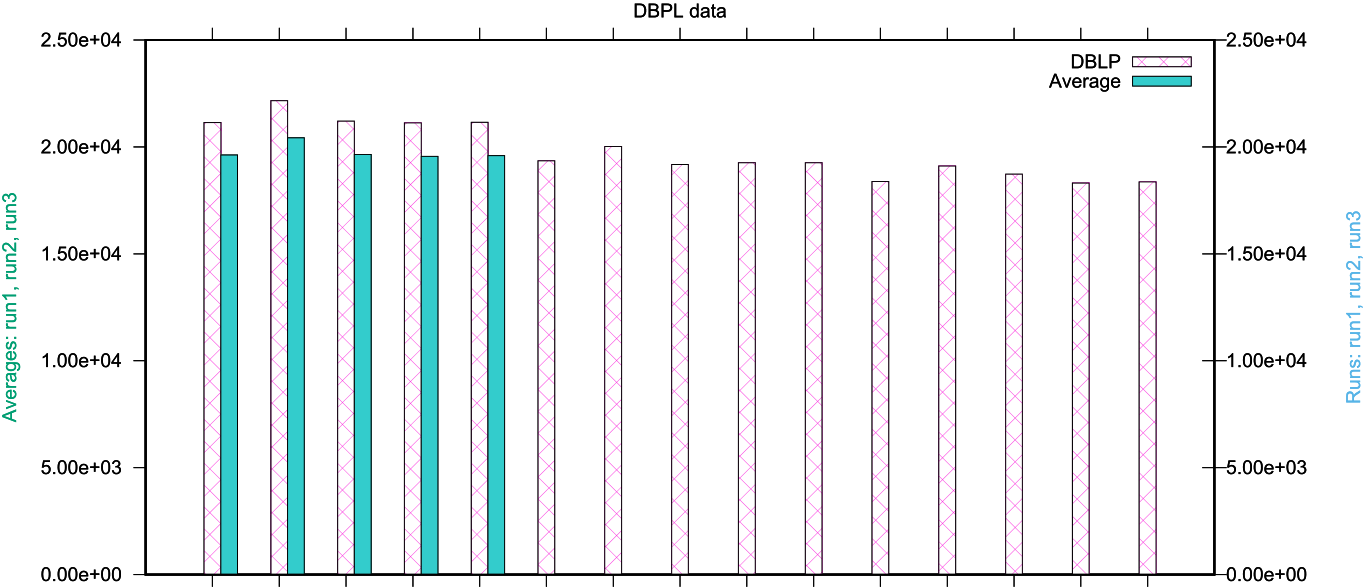






Fig. 4. DBLP data set: True duplicates vs. ME, CT, VT, and FT Algorithms.

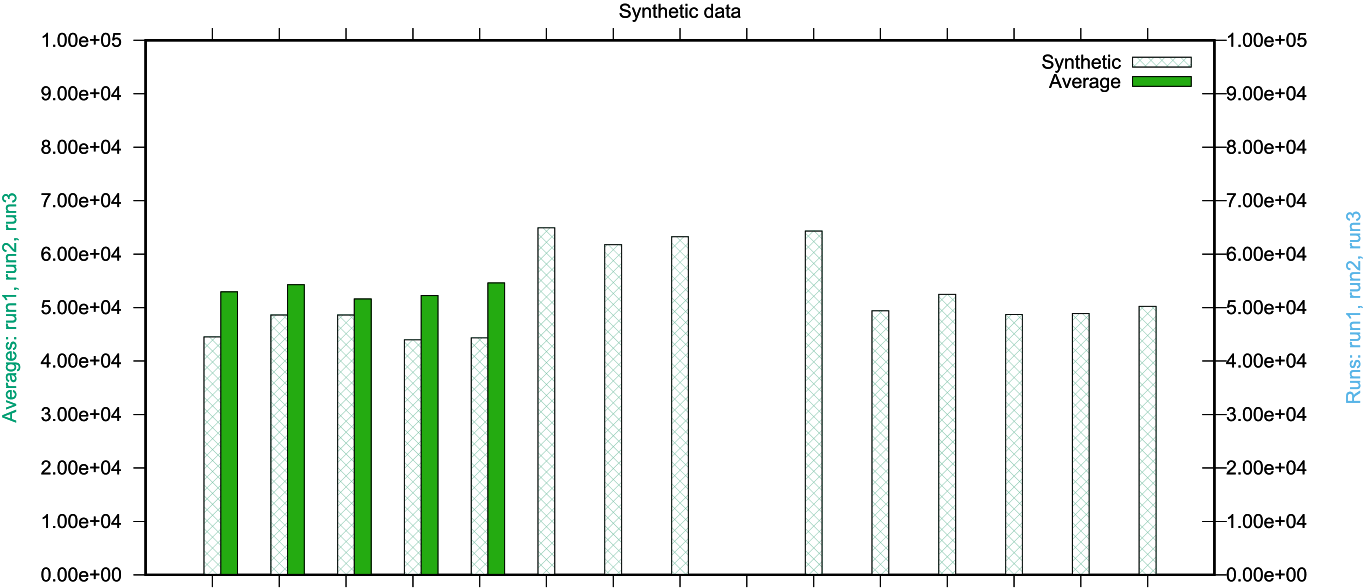




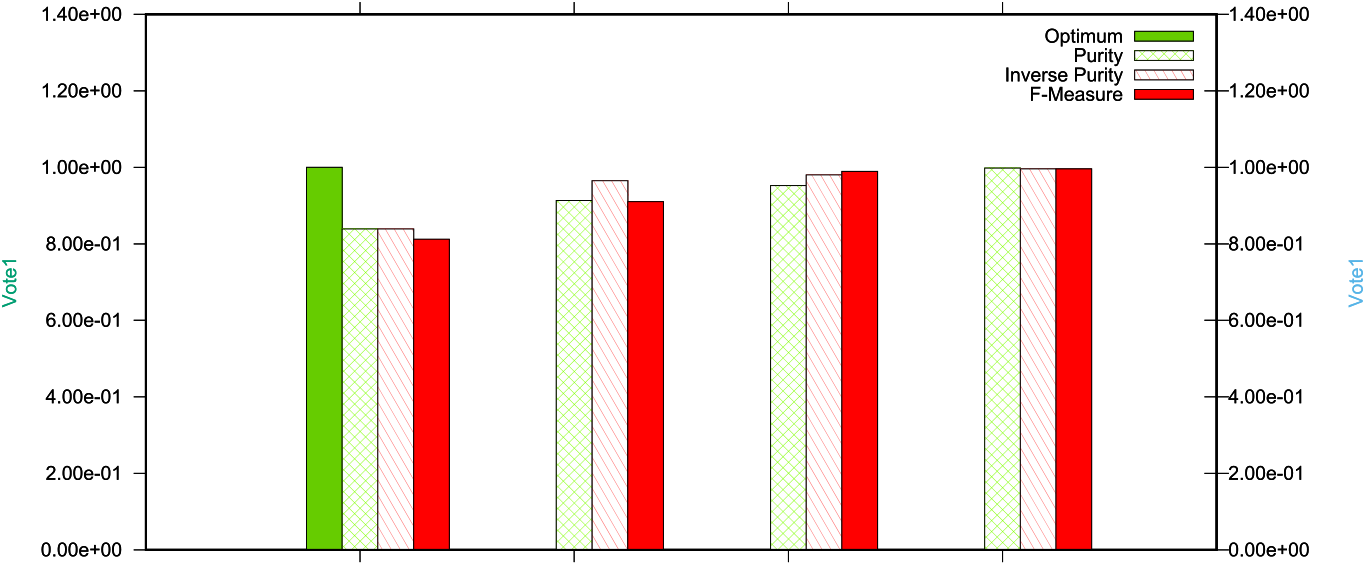


Fig. 5. Synthetic data set: Optimal duplicates vs. ME, CT, VT, and FT Algorithms

12

A. Fellah Array 11 (2021) 100070



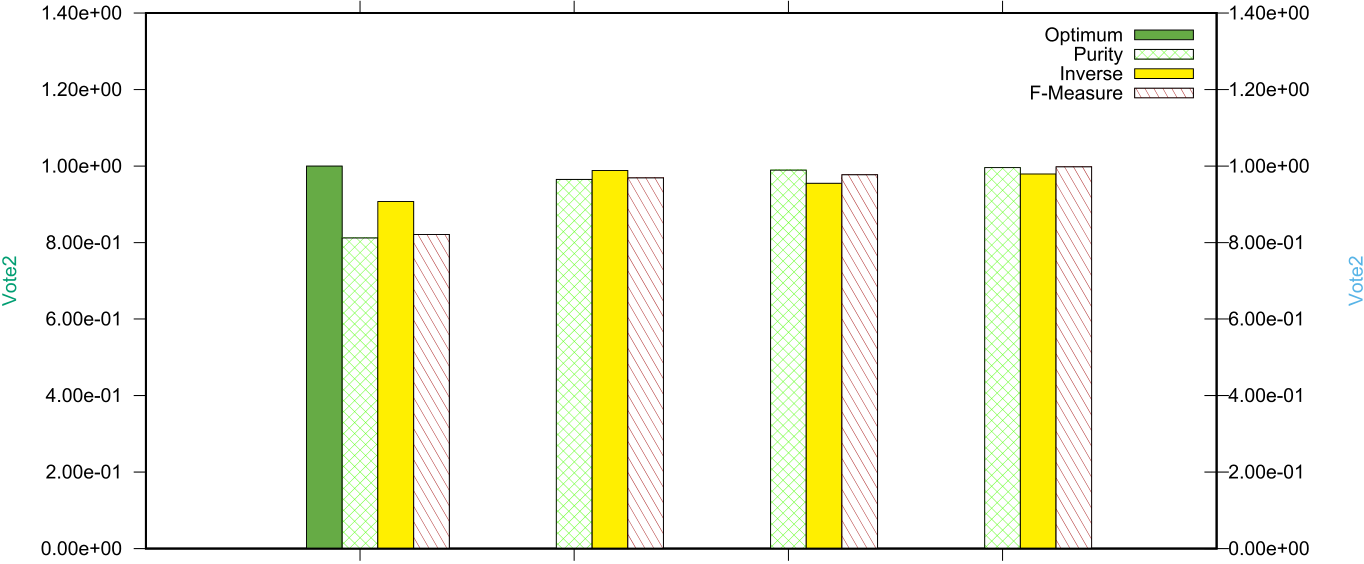


|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |



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



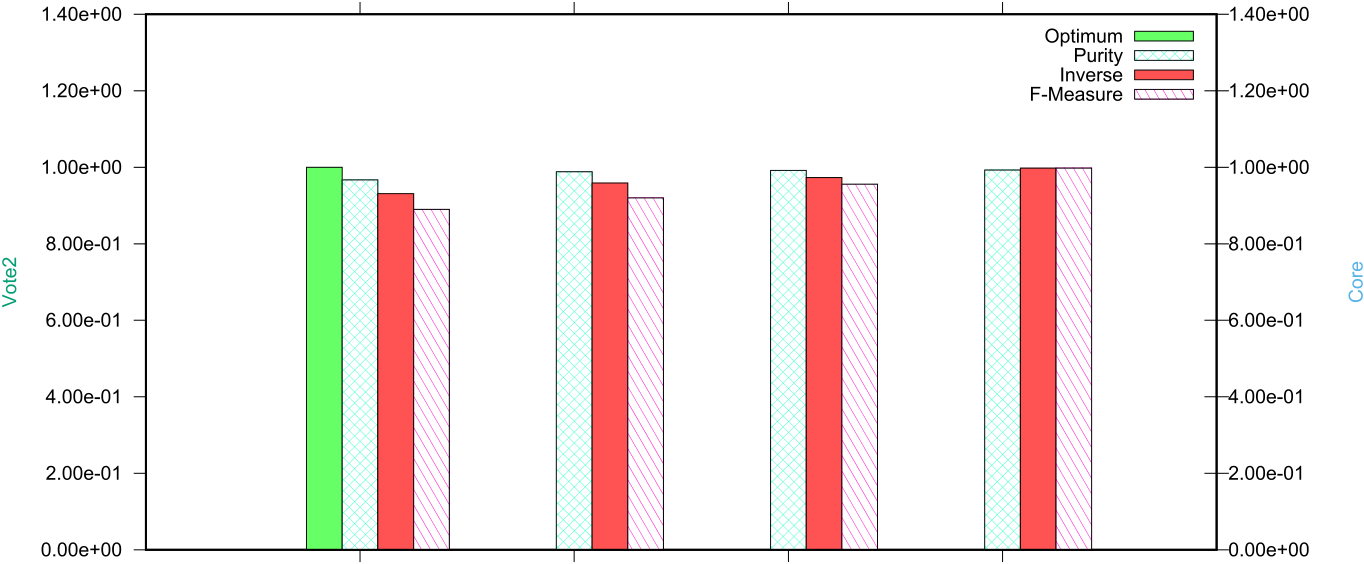


|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |



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





|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

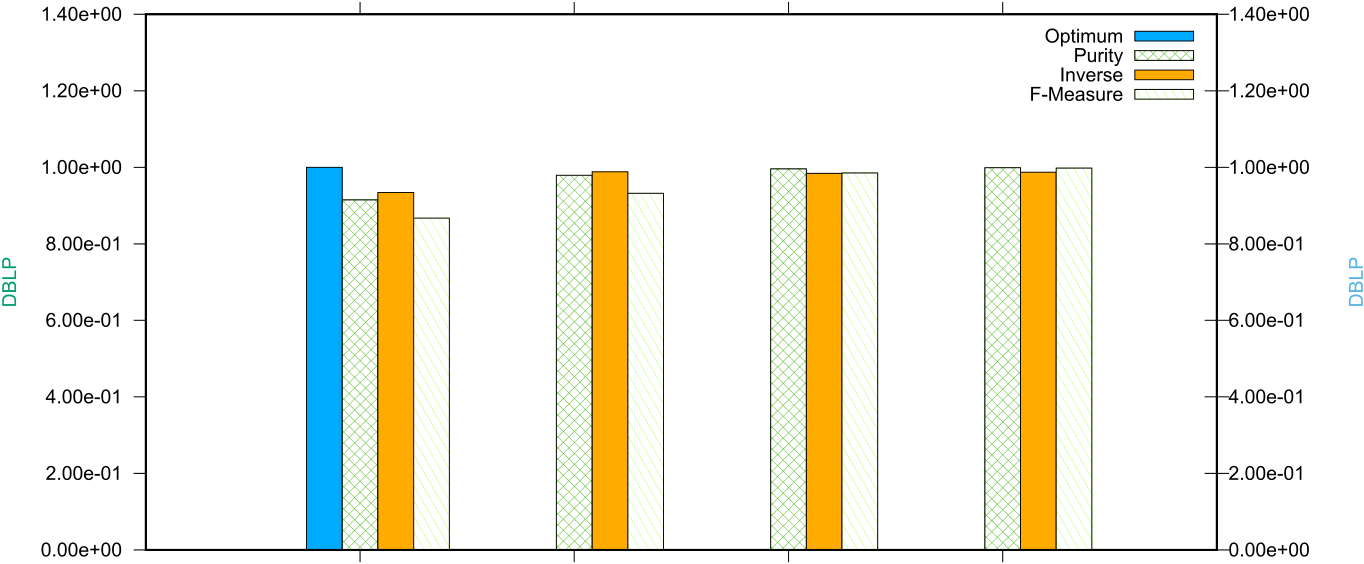


Fig. 8. Core data set: Optimal duplicates vs. ME, CT, VT, and FT measures

13

A. Fellah Array 11 (2021) 100070

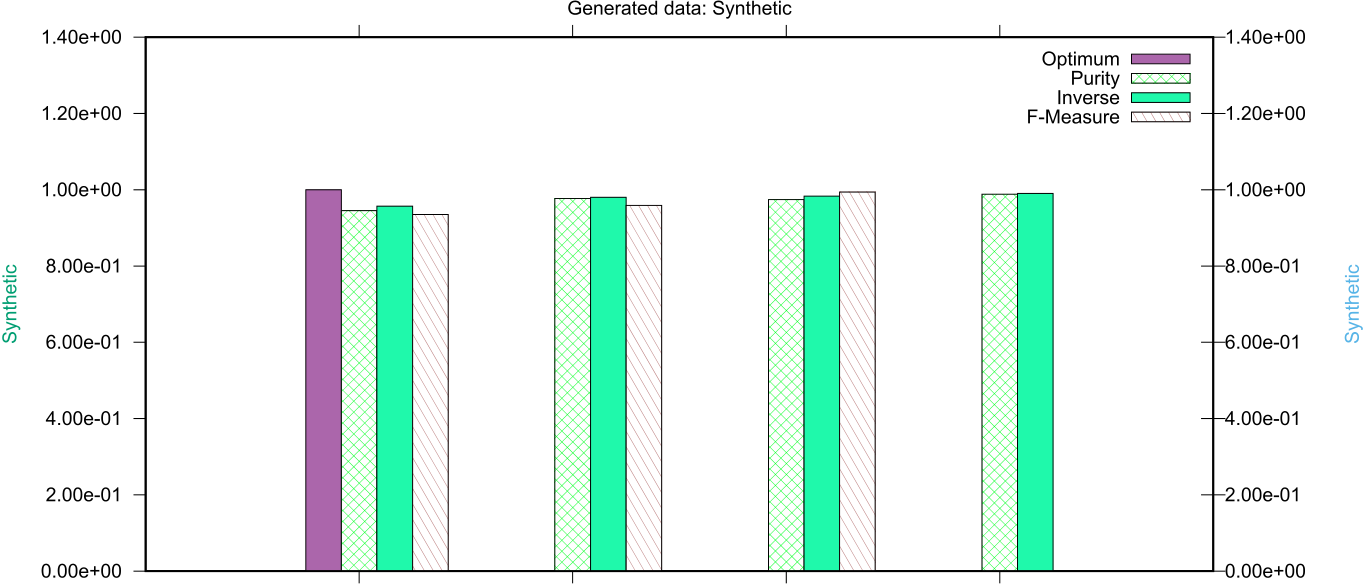




|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |



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

[6] [Weis M, Naumann F, Brosy F. A duplicate detection benchmark for xml (and](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref6)

|  |  |
| --- | --- |
| References | [relational) data. In: Proceedings of the SIGMOD inter. Workshop on information](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref6) [7] [Yan S, Lee D, Kan M, Giles L. Adaptive sorted neighborhood methods for eff](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref7)i[cient](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref7) [quality for information systems. IQIS); 2004. p. 10](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref6)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref7)[9](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref6). |

[1] [Newcombe H, Kennedy J, Axford S, James A. Automatic linkage of vital records.](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref1)

[2] [Christen P. A survey of indexing techniques for scalable record linkage and](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref2) [J Sci 1959;130(3881):954](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref1)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref2)[9](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref1).

[deduplication. IEEE Transactions on Knowledge and Data Engineering (TKDE 2012;](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref2)

[3] [Jaro M. Advances in record-linkage methodology as applied to matching. J Am Stat](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref3) [24(9):1537](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref2)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref3)[55](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref2).

[4] [Monge A. Adaptive detection of approximately duplicate database records and the](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref4) [Assoc 1989;84(406):414](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref3)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref4)[20](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref3).

[database integration approach to information discovery. Ph.D. thesis, Ph.D. Thesis. San Diego: Dept. of Comp. Sci. and Eng., Univ. of California; 1997](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref4).

[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](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref5)–[32](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref5).

14

A. Fellah Array 11 (2021) 100070

[13] [Xiao C, Wang W, Lin X, Yu J, Wang G. Eff](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref13)i[cient similarity joins for near-duplicate](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref13)

[14] [Monge A, Elkan C. Domain-independent algorithm for detecting approximately](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref14) [detection. ACM Trans Database Syst 2011;36(3):15](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref13)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref14)[41](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref13).

[duplicate database records. In: Proceedings of the ACM SIGMOD workshop on](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref14)

[15] [Fellah A, Maamir A. A domain independent methodology for near-duplicate](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref15) [research issues on data mining and knowledge discovery. DMKD); 1997. p. 23](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref14)–9.

[detection. In: Proceedings of the international conference on applied computing,](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref15)

[16] [Navarra G. A guided tour to approximate string matching. ACM Comput Surv 2001;](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref16) [madrid Spain; 2012. p. 139](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref15)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref16)[46](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref15).

[17] D. Moreira, al, Image provenance analysis at scale, IEEE Trans Image Process 27 [33(1):31](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref16)–[88](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref16).

(12).

[18] [Fellah A, Maamir A. Near-optimal domain independent approach for detecting](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref18)

[duplicates. In: Proceedings of the 19th international conference on data mining,](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref18)

[19] [Bharambe D, Jain S, Jain A. A survey: detection of duplicate record. International](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref19) [multimedia and image processing. Paris France; 2017. p. 2633](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref18)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref19)[42](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref18).

[20] [Hassanian-esfahani R, Kargar Mj. Sectional minhash for near-duplicate detection.](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref20) [Journal of Emerging Technology and Advanced Engineering 2012;2(11):298](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref19)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref20)[307](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref19).

[21] [Herschel M, Naumann F, Szott S, Taubert M. Scalable iterative graph duplicate](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref21) [Expert Syst Appl 2018;99(1):203](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref20)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref21)[12](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref20).

[22] [Naumann F, Herschel M. An introduction to duplicate detection. Synthesis Lectures](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref22) [detection. IEEE Trans Knowl Data Eng 2012;4(11). 2294](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref21)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref22)[2108](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref21).

[23] [Winkler W. Overview of record linkage and current research directions. A Statistical](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref23) [on Data Management 2010;2(1):1](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref22)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref23)[87](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref22).

[24] [Winkler W. Approximate string comparator search strategies for very large](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref24) [Research Division, U.S. Census Bureau; 2006. p. 1](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref23)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref24)[44](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref23).

[administrative lists, A Statistical Research Report Series (Statistics 2005-02). U.S.](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref24)

[25] [Baxter R, Christen P, Churches T. A comparison of fast blocking methods for record](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref25) [Census Bureau; 2005. p. 1](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref24)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref25)[9. 02](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref24).

[linkage. In: Proceedings of the ACM SIGKDD workshop on data Cleaning,Record](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref25)

[26] [Monge A. Matching algorithms within a duplicate detection system. IEEE Data](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref26) [linkage, and object consolidation; 2003. p. 25](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref25)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref26)[8](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref25).

[27] [Yandrapally R, Stocco A, Mesbah A. Near-duplicate detection in web app model](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref27) [Engineering Bulletin 2000;23(4):14](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref26)[–](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref27)[20](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref26).

[inference. In: Proceedings of the ACM/IEEE 42nd international conference on](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref27)

[software engineering; 2020. p. 186](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref27)–[97](http://refhub.elsevier.com/S2590-0056(21)00018-7/sref27).

15