Transforming Pairwise Duplicates to Entity Clusters for High-quality Duplicate Detection

UWE DRAISBACH, Hasso-Plattner-Institute, University of Potsdam, Germany
PETER CHRISTEN, Australian National University, Australia
FELIX NAUMANN, Hasso-Plattner-Institute, University of Potsdam, Germany

Duplicate detection algorithms produce clusters of database records, each cluster representing a single real-world entity. As most of these algorithms use pairwise comparisons, the resulting (transitive) clusters can be inconsistent: Not all records within a cluster are sufficiently similar to be classified as duplicate. Thus, one of many subsequent clustering algorithms can further improve the result.

We explain in detail, compare, and evaluate many of these algorithms and introduce three new clustering algorithms in the specific context of duplicate detection. Two of our three new algorithms use the structure of the input graph to create consistent clusters. Our third algorithm, and many other clustering algorithms, focus on the edge weights, instead. For evaluation, in contrast to related work, we experiment on true real-world datasets, and in addition examine in great detail various pair-selection strategies used in practice. While no overall winner emerges, we are able to identify best approaches for different situations. In scenarios with larger clusters, our proposed algorithm, Extended Maximum Clique Clustering (EMCC), and Markov Clustering show the best results. EMCC especially outperforms Markov Clustering regarding the precision of the results and additionally has the advantage that it can also be used in scenarios where edge weights are not available.

CCS Concepts: • **Information systems** → **Deduplication**; **Entity resolution**; *Data cleaning*; *Clustering*; *Clustering and classification*;

Additional Key Words and Phrases: Record linkage, data matching, entity resolution, deduplication, clustering

ACM Reference format:
Uwe Draisbach, Peter Christen, and Felix Naumann. 2019. Transforming Pairwise Duplicates to Entity Clusters for High-quality Duplicate Detection. *J. Data and Information Quality* 12, 1, Article 3 (December 2019), 30 pages.
https://doi.org/10.1145/3352591

1 CLUSTERS OF DUPLICATES

Duplicate detection is the process of finding multiple records in a dataset that represent the same real-world entity [13, 26]. It is also known as entity matching, entity resolution, data matching, record linkage, reference reconciliation, and many other terms. Duplicate detection has a high

Authors' addresses: U. Draisbach and F. Naumann, Hasso-Plattner-Institut für Digital Engineering gGmbH, University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany; emails: {uwe.draisbach, felix.naumann}@hpi.de; P. Christen, The Australian National University, Canberra, ACT 2600, Australia; email: peter.christen@anu.edu.au.

1936-1955/2019/12-ART3 $15.00
https://doi.org/10.1145/3352591

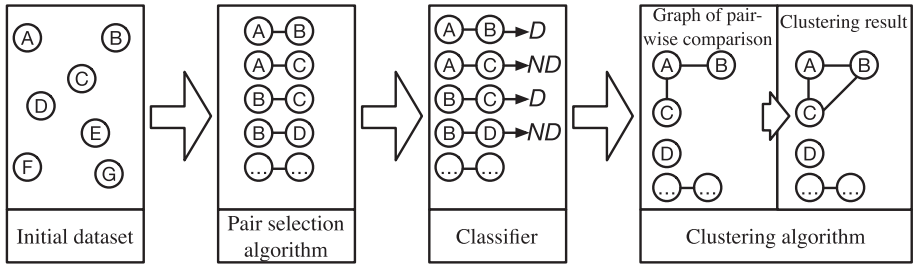


Fig. 1. Illustration of the duplicate detection process.

relevance in many areas, such as master data management, customer relationship management, and data integration (e.g., data warehousing). Its challenge is the missing key to identify multiple records representing the same entity. Rather, we deal with fuzzy duplicates: Records with similar attribute values should be classified as duplicates, whereas different values are an indicator of non-duplicates.

A variety of similarity measures have been proposed in recent years [8, 13, 26]. They all have in common that their results might be erroneous. Records representing the same entity might be assigned a low similarity and, vice versa, records representing different entities might have been assigned a high similarity value (e.g., twins). Deciding whether a record pair is a duplicate or not is a challenging task even for humans, and sometimes no clear decision can be reached. For example, given two records with the same first and last name of a person, but totally different addresses, it is difficult to decide whether these records represent two different persons or one person that has moved to a new address. Additional information might be necessary, but is often not available in a dataset.

There are three main algorithmic challenges for duplicate detection. First, we need effective algorithms that support the classification of record pairs as duplicate or not, even if the records are incomplete or erroneous. Second, the increasing size of databases leads to the necessity for efficient algorithms that select candidate pairs from the full pair-wise comparison space. An exhaustive comparison is in most real-world scenarios not a feasible approach [7]. This increases the problem of uncertainty, as we do not have a complete picture on the relations between all records. Finally, we need algorithms that create clusters, in which each record represents the same real-world entity, although the record pair classifications might be incomplete or erroneous.

The process of duplicate detection is typically divided into three steps, which are illustrated in Figure 1. First, a pair selection algorithm selects a set of candidate pairs from the initial dataset or datasets (in case of linking multiple datasets). Due to the large workload for an exhaustive comparison, usually only candidate pairs are selected that have a high chance of being duplicates [7, 26]. Second, a classifier is used to decide for each record pair whether it represents the same real-world entity or not. For this step, similarity functions, such as Levenshtein or Jaro-Winkler [8], are used to calculate attribute similarities that are aggregated to an overall similarity. A threshold can then be used to decide whether a record pair is a duplicate (*D*) or a non-duplicate (*ND*). Other possibilities are machine learning or a rule-based classification, which scans a set of rules until one rule finally classifies a record pair as duplicate or non-duplicate. In the third and final step, the transitive closure is calculated to obtain a transitively closed result set. This final step considers only positive duplicate decisions and creates additional duplicate pairs. This last step can be the reason of many errors and is in the focus of this article.

Calculating the transitive closure disregards negative classifications (the similarity of a record pair is below the threshold and thus classified as non-duplicate). Table 1 shows sample records

Table 1. FreeDB Example

ID	Artist	Disc	Genre	Year
1	Nora Roberts	Angels Fall	Audio Book	2006
2	Nora Roberts	Angels Fall-Disc04	Audiobook	2006
3	Nora Roberts	Angels Fall-Disc05	Audiobook	2006
4	Nora Roberts	Angels Fall-Disc06	Audiobook	2006
5	Nora Roberts	Angels Fall-Disc07	Audiobook	2006

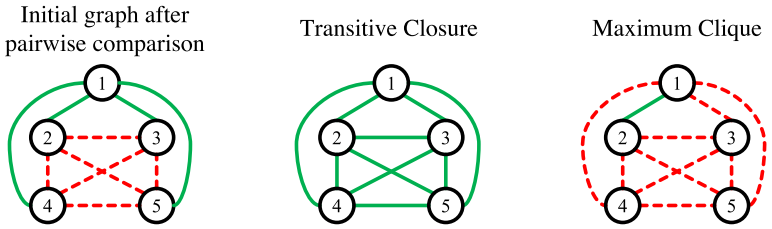


Fig. 2. Clustering of the example in Table 1, showing (i) the initial pairwise comparison graph, (ii) result of the transitive closure, which assigns all elements to the same cluster, and (iii) the maximum clique approach that creates one cluster with two elements and three singleton clusters.

extracted from freeDB.[1] As we can see, for all records the artist and the year are the same. The values for *disc* and *genre* have only small differences, which leads to a high string similarity for any pair of these records. But for example disc 4 of an audio book is not the same real-world entity as disc 5 of the same audio book. In a real application, the similarity function would be refined, e.g., by implementing a rule that independently of the record pair similarity classifies a record pair as non-duplicate if these are discs in a multiple CD set. Such rules are domain dependent, so for books there might be other rules necessary. Unfortunately, the rule fails for record 1, which does not contain any information about the disc. As a result, record 1 would possibly be classified as duplicate of all other records and thus connect them. This example shows the challenges of duplicate detection and the possibly negative impact of calculating the transitive closure.

Figure 2 illustrates this problem. After the pairwise comparison, we initially have a connection between record 1 and records 2–5, with records 2–5 being classified as non-duplicates. Calculating the transitive closure ignores that records 2–5 are classified as non-duplicates: all records are now in the same cluster, which means they represent the same real-world entity.

The opposite approach would be to reclassify duplicates as non-duplicates, until we reach a maximum clique. For our example in Figure 2, we could first reclassify edge $\langle 1, 3\rangle$ as non-duplicate, then edge $\langle 1, 4\rangle$ and finally edge $\langle 1, 5\rangle$. We now have four cliques $\{1, 2\}, \{3\}, \{4\}, \{5\}$. Note that the reclassified edges are selected arbitrarily. Edge $\langle 1, 2\rangle$ might also be reclassified instead of one of the other edges.

The result of a misclassification is a reduction in quality of the duplicate detection result. In this example, only three pairs are affected, which might be either false negatives that reduce the recall (completeness of the result) or false positives that reduce the precision (correctness of the result). But for other datasets with larger clusters, misclassification has a high impact on the overall result. One often cited dataset to evaluate duplicate detection results is the CORA dataset.[2] Some issues

[1] freedb is a database to look up CD information: http://www.freedb.org/.
[2] http://www.cs.umass.edu/~mccallum/data.html.

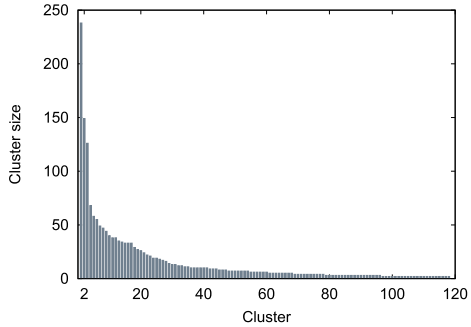


Fig. 3. Cluster sizes of the CORA dataset for clusters with at least two records. Due to some very large clusters, a false connection between two clusters might result in a high number of false positives.

with this dataset and the creation of a gold standard are described in Reference [11]. In particular, it contains clusters in a wide range of sizes.

Figure 3 shows the cluster sizes in decreasing order for the CORA dataset. The two largest clusters have 238 and 149 records, respectively. A single misclassification of a record pair with one element from each cluster leads to $238 * 149 = 35{,}462$ misclassified record pairs after calculating the transitive closure. Note that this example uses a pairwise measure, because this is used in most duplicate detection publications. There are also other cluster-based measures, e.g., the Generalized Merge Distance, that can also be used to evaluate duplicate detection results. Menestrina et al. describe different measures to evaluate duplicate detection result [24].

The goal of this article is the development and evaluation of several known and new clustering algorithms for duplicate detection. This input for these clustering algorithms is the result of a possibly incomplete and inconsistent pairwise comparison of all records. The input can be described as a graph and the clustering algorithms use a re-classification of single edges to create cliques. The elements in each clique represent the same real-world entity. The contributions of our article are:

- Presentation of three new clustering algorithms. The first two new algorithms use the structure of the input graph and thus are, in contrast to the third new and many other existing clustering algorithms, not dependent on edge weights.
- Detailed presentation of several existing clustering algorithms that belong to the best clustering algorithms in the context of duplicate detection, as evaluated in References [19, 36].
- Comprehensive experimental evaluation of all algorithms using (i) real-world datasets and (ii) different pair-selection strategies.

In Section 2, we formalize the problem of clustering duplicate detection result. Section 3 describes a new clustering approach that does not depend on edge weights, whereas in Section 4, we present a new algorithm that uses edge weights for classification. Section 5 gives an overview of existing clustering algorithms that are evaluated in Section 6 with several datasets, followed by an overview of more general related work in Section 7 and a conclusion of our work in Section 8.

2 PROBLEM DESCRIPTION

Duplicate detection is the process of finding objects that represent the same real-world entity. Given a set of records $R = \{r_1, \ldots, r_n\}$, a pair selection algorithm creates candidate pairs $<r_i, r_j>$ of records that are classified as duplicate D or as non-duplicate ND. The result of a pairwise record comparison is an undirected Graph $G = \{V, E\}$ with V as the set of vertices and E as the set of labeled edges.

ID	Name	City	Entity	Sort.	Block
1	Doe, John	Munich	E1	4	B1
2	Doan, J.	Berlin	E2	2	B2
3	Do, John	Berlin	E1	1	B2
4	Doe, Jhon	Berlin	E1	3	B2
5	Poe, Jan	Munich	E3	6	B1
6	Po, Jan	Munich	E3	5	B1

(a) Example records.

ID	**1**	**2**	**3**	**4**	**5**	**6**
1	0	5	1	1	3	4
2		0	5	5	5	5
3			0	2	4	3
4				0	3	4
5					0	1
6						0

(b) Edit distance for sample records.

Fig. 4. Example records and their pairwise edit distance.

Depending on the pair selection algorithm, the graph can but does not necessarily have to be complete. As for most pair selection algorithms, not all record pairs are classified—edges between some vertices might be missing. Especially for large datasets it is unlikely that we have a complete graph, because of the quadratic total number of comparisons blocking would have to be applied [7].

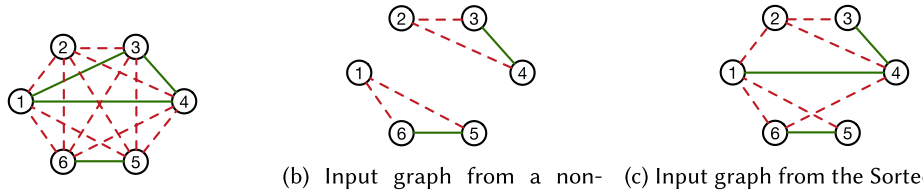
Depending on the classifier, the edges might be weighted or just classified as duplicate or non-duplicate. If weighted, then each edge has a weight w that represents the similarity of the records, with $0 \leq w \leq 1$. A high weight means a high similarity of the records. Record pairs with a similarity higher than a threshold t are classified as duplicates, whereas pairs with a lower threshold are classified as non-duplicates. As we examine in Section 5, some clustering algorithms require a weighted graph. If the classifier is not based on a similarity function, then we can also use $w = 1.0$ for duplicates and $w = 0.0$ for non-duplicates.

The subgraph induced by all edges with $w \geq t$ comprises between 1 and n connected components, with $n \leq |V|$. In each component, we have a path from every vertex to all other vertices in the component with a "is-duplicate-of" relation. But for most classifiers the relation is not transitive, i.e., $\langle r_1, r_2 \rangle = D$ and $\langle r_2, r_3 \rangle = D$ does not imply $\langle r_1, r_3 \rangle = D$ [3]. Transitive classifiers are discussed in [25]. Thus, the components are not necessarily cliques.

The objective of the clustering algorithm is to change the classification of edges to create a set of disjoint cliques $C = \{c_1, \ldots, c_k\}$, with $c_i \subseteq R$ and $c_i \cap c_j = \emptyset$, so that each clique c_i is a set of records that are assumed to represent the same real-world entity. We call a change of an edge classification an *edge switch*. Each clique c_i can be a singleton or contain multiple records and it holds $c_1 \cup c_2 \cup \cdots \cup c_k = R$. The challenge for the clustering algorithm is that it neither knows which edges have a wrong classification, nor the correct size that each clique should have. The knowledge of the clustering algorithm is restricted to the structure of the subgraph, and, in case that it is a weighted subgraph, the weights of the edges. The latter information can help to decide which edge classification should be switched. For example, if we have $t = 0.7$ and two edges e_1 and e_2 with $w_1 = 0.71$ and $w_2 = 0.99$, then it might be a good choice to change the classification of e_1 from duplicate to non-duplicate, as the weight is just slightly above the threshold.

Another challenge for the clustering algorithm is that the input graph is not necessarily complete. Figure 4(a) shows six sample records. If we have a classifier that calculates the edit distance (shown in Figure 4(b)) for the name and classifies a record pair as duplicate if the edit distance ed is $ed \leq 2$, then we have three clusters $\{1, 3, 4\}, \{2\}, \{5, 6\}$. Depending on the used pair selection algorithm, we have different input graphs for the clustering step as shown in Figure 5. A solid line edge represents a duplicate, whereas a dotted line edge represents a non-duplicate.

Note that there is not an edge between all vertices. This would only be the case for an exhaustive comparison, as shown in Figure 5(a). Most duplicate detection algorithms select only a subset of candidate pairs for classification to reduce costs. Thus, the classification of some record pairs is



(a) Input graph from an exhaustive comparison.

(b) Input graph from a non-overlapping blocking algorithm with city as blocking key.

(c) Input graph from the Sorted Neighborhood Method [25] with name as sorting key and window size 3.

Fig. 5. Impact of the pair selection algorithm on the clustering input graph for the example in Figure 4.

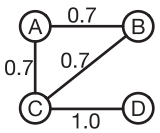


Fig. 6. Sample graph after pairwise classification.

unknown and therefore we do not have an edge between all vertices. In Figure 5(b), we see the input graph of disjoint blocking [7] with city as blocking criteria. Record pairs were only classified if they are in the same block. Figure 5(c), however, shows the result of the Sorted Neighborhood method [25]. Both pair selection strategies select only a subset of record pairs and thus make it more complicated for the clustering algorithm to achieve good results.

Given a set of records as nodes and a set of weighted edges between the records, a clustering algorithm should produce a set of clusters, where (i) each cluster contains only records that represent the same real-world entity and (ii) the number of clusters is equal to the number of real-world entities in the dataset. The clustering algorithm can, on the one hand, create missing edges and, on the other hand, reverse the classification of existing edges to fulfill these two goals.

In the output graph of the clustering algorithm, all records representing the same real-world entity are in the same maximal clique with only edges labeled as *D*, while between two maximal cliques there are only edges labeled as *ND*. Each maximal clique is then a cluster of elements representing the same real-world entity. This means that if we remove all *ND* edges, the resulting graph contains a set of connected components in which each component is a clique.

3 MAXIMUM CLIQUE CLUSTERING

In this section, we describe two novel clustering approaches that make use of the the structure of the resulting graph of the pairwise classification. The idea of these clustering approaches is that the structure of the input graph is more important than the edge weights. An edge with a similarity just a little above the threshold is as important for the clustering as an edge with a similarity of 1.0. More important are further edges that support the classification.

Figure 6 shows a sample graph representing the result of a pairwise classification, in which edges with a similarity ≥ 0.7 were classified as duplicates. The similarity of edges $\langle A, B\rangle$, $\langle B, C\rangle$, and $\langle B, C\rangle$ is just high enough to be classified as duplicate, whereas $\langle C, D\rangle$ has a much higher similarity. But for example edge $\langle A, C\rangle$ is supported by the edges $\langle A, B\rangle$ and $\langle B, C\rangle$, because all three edges confirm that elements A, B, C represent the same entity. The edge $\langle C, D\rangle$ is not supported by other edges, because neither $\langle A, D\rangle$ nor $\langle B, D\rangle$ were classified as duplicate. In Section 5, we present other clustering algorithms that have a focus on the edge weights and would rather assign $\langle C, D\rangle$ to the same cluster.

3.1 Maximum Clique Clustering (MCC)

The MCC algorithm consists of two steps: First, it calculates for a component the maximum clique. The maximum clique is the biggest maximal clique (a clique is maximal if it cannot be extended by another vertex of a graph). We use the Bron-Kerbosh algorithm [5] to calculate the maximal cliques. The maximum clique is the first cluster of the component and in the second step its vertices are removed from the component. Thus, the component might be split into several smaller components, each being a maximum clique. For all remaining components the algorithm is repeated until all vertices are assigned to a cluster. Note that the first step is not necessary for components with one or two vertices only, as in these cases the component itself is already a maximum clique.

3.2 Extended Maximum Clique Clustering (EMCC)

The extended maximum clique clustering is an extension of MCC. It is especially useful if we have near-cliques, e.g., just a single or only a few edges are missing to increase a clique. As for MCC, we start calculating the maximum clique for a component. If there are multiple possible maximum cliques, then we have to select one. We have evaluated three strategies:

(1) Select arbitrarily the first maximum clique.
(2) Select the maximum clique with the most edges to vertices that are not in the maximum clique. The hypothesis is that a maximum clique with many edges to outside vertices can more likely be extended.
(3) Select the maximum clique with the fewest edges to vertices that are not in the maximum clique. The idea of this approach is that only a small number of edges in the component need to be deleted, if the maximum clique cannot be extended anymore.

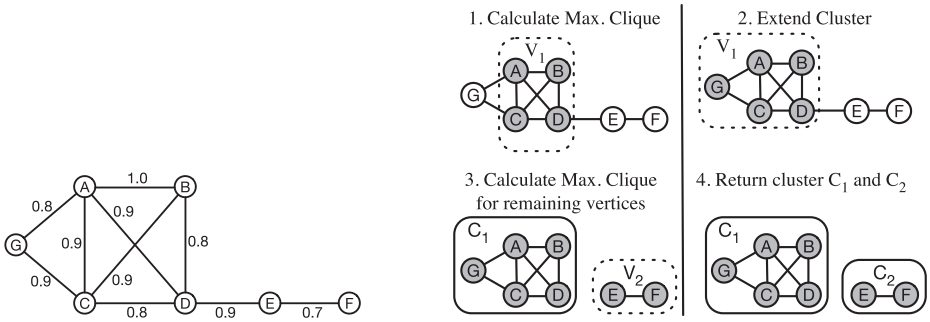
We gained the best results with the second approach, selecting the maximum clique with the most edges to vertices that are not in the maximum clique.

In the second step, we iteratively try to increase the selected clique with further vertices. If a vertex that is not in the maximum clique fulfills a specific condition, then we suppose that it also represents the same real-world entity as the vertices in the maximum clique. Note that if we add these vertices, we do not have a clique anymore, so we now call it a cluster. We evaluated two different conditions for adding a vertex:

(1) The vertex needs an edge to a specific percentage of vertices in the cluster. Due to the increase of vertices in the cluster in each iteration, the number of required edges is also rising.
(2) Like the first approach, but additionally a vertex needs a specific percentage of edges to the vertices in the maximum clique. As with each iteration the cluster is increasing, this additional constraint prevents that vertices are added that have no or only a few edges to the original maximum clique.

Our experiments showed that the second condition is not necessary to obtain better results. Thus, we use only the first condition for EMCC. The required percentage is represented by a parameter τ. This extending step is repeated iteratively until no further vertex can be added. In each iteration the number of vertices in the cluster is increased and thus also the number of required edges to the cluster is increased. Algorithm 1 shows the pseudocode for EMCC.

Figure 7 shows another sample of a pairwise comparison and the calculation of EMCC with $\tau = 0.5$. After calculating the first maximum clique, we have $V_1 = \{A, B, C, D\}$. Thus, in the first iteration only vertices with at least two edges to V_1 can be added to the cluster. In the second iteration, V_1 contains five vertices, so at least three edges are required to extend V_1. As we cannot extend



(a) Sample result of pairwise comparison. (b) EMCC for the sample in Fig. 7a with $\tau = 0.5$.

Fig. 7. EMCC clustering.

ALGORITHM 1: Extended Maximum Clique Clustering (EMCC)

Input: A set $C = \{c_1, c_2, \ldots, c_x\}$ of undirected connected components (V, E).
A threshold τ for the clique extension.
Output: A set of clusters RC in which each cluster represents a real-world entity.

```
1  RC ← ∅;                                                    // Result set of clusters
2  foreach c_x ∈ C do                                         // Iterate over connected components
3      V ← ∅;                                                 // Set of vertices
4      while c_x ≠ ∅ do
5          V ← MaxClique(c_x) with Max(|E|);
6          do                                                 // Check if extension is possible
7              if ∃v(|E(v, v_y) with v ∈ c_x \ V, v_y ∈ V|/|V| ≥ τ) then
8                  V ← V ∪ {v with Max(|E(v, v_y) with v ∈ c_x \ V, v_y ∈ V|)}
9          while V has been extended;
10         RC ← new cluster with vertices in V;
11         c_x ← c_x \ V;                                     // Remove elements in cluster from c_x
12 return RC;
```

V_1 anymore, the vertices of V_1 represent the first cluster and are removed from the connected component. For the remaining vertices E and F, we calculate again the maximum clique. As there are no further vertices left to extend $V_2 = \{E, F\}$, EMCC returns two clusters: $\{A, B, C, D, G\}$ and $\{E, F\}$.

Please note the impact of parameter τ on the overall clustering result. In the example, with $\tau > 0.5$ EMCC would not add vertex G to the first maximum clique. Thus, EMCC would give the same clustering result as algorithm *Maximum Clique Clustering*.

The choice of τ is important to obtain good quality clustering results. The two extreme cases are $\tau = 0.0$ and $\tau = 1.0$. In the case of $\tau = 0.0$, we are adding all vertices of a connected component to the same cluster. Thus, the clustering result equals the result of the Transitive Closure. On the contrary, with $\tau = 1.0$, we are not adding any additional vertex to the maximum clique, and we get the same result as for MCC. Our experiments showed that in most cases there is not just a single optimal threshold, but rather a threshold range. Furthermore, even the selection of τ close to the best range showed only a very small negative impact on the overall clustering result.

The extension of the maximum clique in EMCC is similar to the creation of *δ-cliques* in GCluster [36], which we describe in Section 5.2. The main difference between these two algorithms is that GCluster tries to maximize the edge weights within a cluster, whereas EMCC is more concerned

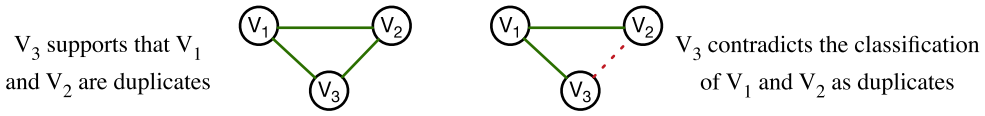


Fig. 8. Additional vertices can support or contradict the classification of a record pair as a duplicate. Dotted line edges represent a classification as non-duplicate and solid line edges as duplicates.

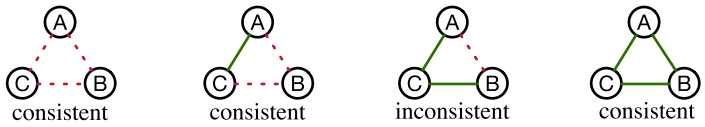


Fig. 9. GECG: consistent and inconsistent triangles.

Table 2. Number of Triangles in a Connected Component

v	1	2	3	4	5	6	7	8	9	10
t	0	0	1	4	10	20	35	56	84	120

about the structure of a cluster. EMCC can also be used in scenarios where no similarity values for edges are available, but only classifications as duplicate or non-duplicate.

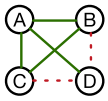
4 GLOBAL EDGE CONSISTENCY GAIN (GECG)

GECG is also a novel clustering approach and it is based on the idea that for each edge we can check whether the classification as duplicate or non-duplicate is correct by taking further edges into account. Assume we have an edge $e_{v1,v2}$ for vertices v_1 and v_2, and the edge is classified as duplicate. Additionally, we have a third vertex v_3 and edges $e_{v1,v3}$ and $e_{v2,v3}$ are also classified as duplicate, then v_3 supports that v_1 and v_2 are really duplicates. However, if only $e_{v1,v3}$ is classified as duplicate, and $e_{v2,v3}$ is classified as non-duplicate, then v_3 contradicts the classification of $e_{v1,v2}$ as duplicate, as shown in Figure 8.

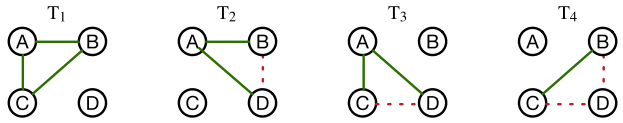
GECG tries to increase the consistency of a connected component. To measure the consistency of a connected component, GECG considers all possible triangles, i.e., all possible sets with three vertices. The vertices in a triangle are consistent, if no edge, one edge, or all edges are classified as duplicate. If a triangle has no duplicate edge, then the vertices are not connected and represent three different real-world entities. With only one duplicate edge, we have a cluster of two vertices and a singleton. If all three edges represent a "is-duplicate-of" relation, then all three vertices represent the same real-world entity. Only if we have two edges classified as duplicate and one edge classified as non-duplicate is this triangle inconsistent. In this case, we can switch any edge in this triangle to make it consistent. An edge switch either results in a pair and a singleton or a cluster of all three vertices. Figure 9 shows the possible triangles and their classification as consistent and inconsistent.

The authors of Reference [15] also define incomplete triangles, if two edges are classified as duplicate and one edge is unknown (e.g., the pair selection algorithm has only created two of the possible three record pairs). We do not consider this case, but implicitly assume that unknown edges are classified as non-duplicate.

The number of triangles in a set of vertices can be calculated with the binomial coefficient $\binom{v}{3} = v * (v-1) * (v-2)/6$. Table 2 shows the number of triangles t for connected components with v vertices.



(a) Connected component from pairwise comparison.



(b) Triangles in the connected component of Fig. 10a.

Fig. 10. Connected component from a pairwise comparison and the triangles that are considered by GECG.

Table 3. Edge Switch Result for the Example in Figure 10

Switched Edge	Consistant triangle T_1	T_2	T_3	T_4	# cons. triangles	Cons. gain
Initial state	yes	no	no	yes	2	-
$E_{A,B}$: $D \rightarrow ND$	no	yes	no	yes	2	0
$E_{A,C}$: $D \rightarrow ND$	no	no	yes	yes	2	0
$E_{A,D}$: $D \rightarrow ND$	yes	yes	yes	yes	4	2
$E_{B,C}$: $D \rightarrow ND$	no	no	no	yes	1	−1
$E_{B,D}$: $ND \rightarrow D$	yes	yes	no	no	2	0
$E_{C,D}$: $ND \rightarrow D$	yes	no	yes	no	2	0

GECG first identifies the number of inconsistent triangles. For each edge, it calculates the consistency gain if the classification of the edge is switched, i.e., the edge classification is switched from duplicate to non-duplicate or vice versa. The consistency gain is defined as the number of consistent triangles after the edge switch minus the number of consistent triangles before the edge switch. Figure 10 gives an example. We have a connected component (Figure 10(a)) with four vertices, which results into four triangles T_1 to T_4 (Figure 10(b)). Before GECG is executed, only triangles T_1 and T_4 are consistent.

Table 3 shows for each edge the result of an edge switch. Initially, T_1 and T_4 are consistent triangles. The consistency gain is only positive, if we switch edge $E_{A,D}$ from duplicate to non-duplicate.

This step of calculating the information gain for each edge and switching the edge with the highest consistency gain is repeated until we get a clique (or a singleton). In this case, we remove the vertices of the clique from the connected component and return them as the next cluster, representing a real-world entity. For the remaining vertices, we repeat the step of switching edges. An edge is only switched, if the consistency gain is positive (>0). If no edge can be switched with a positive consistency gain, then all remaining vertices are added to the same cluster, representing the same real-world entity. In case we have multiple edges with the same consistency gain, we select the edge with $min(||threshold - similarity||)$.

Algorithm 2 shows the pseudocode of GECG. GECG iterates over all connected components (line 3). As long as there exists an inconsistent triangle in the selected component and $max_{cg} > 0$ (line 7), GECG recalculates for all edges of the selected connected component the consistency gain for an edge switch and determines the maximum consistency gain (line 8). If the consistency gain is positive (line 9), then one of these edges with the maximum consistency gain is selected (line 12 or 14) and then switched (line 15). In case that the edge switch splits the connected component, GECG continues with the larger component and adds the smaller component to the set of connected components that still have to be processed (lines 16–18). If there are no more inconsistent triangles in

ALGORITHM 2: Global edge consistency gain (GECG)

Input: A set $C = \{c_1, c_2, \ldots, c_x\}$ of undirected connected components (V, E).
A threshold τ used for a pairwise classification.
Output: A set of clusters RC in which each cluster represents a real-world entity.

1 $RC \leftarrow \emptyset$
2 $max_{cg} \leftarrow 1$ // Max. consistency gain
3 **foreach** $c_x \in C$ **do** // Iterate over all connected components
4 $c \leftarrow c_x \in C$ // Select next component to be processed
5 $C \leftarrow C \setminus \{c_x\}$
6 **while** *c contains inconsistent triangle with*
7 $e_{ij} \geq \tau, e_{jk} \geq \tau,$ *and* $e_{ik} < \tau$ **and** $max_{cg} > 0$ **do** // Iteratively process selected component
8 $max_{cg} \leftarrow max\{get\_cg(e,c) : e \in c\}$ // Calculate max. consistency gain
9 **if** $max_{cg} > 0$ **then** // Check if max. consist. gain > 0
10 $E_{MaxCG} \leftarrow \{e \in c : get\_cg(e, c) = max_{cg}\}$ // Get edges with max. consistency gain
11 **if** $\|E_{MaxCG}\| = 1$ **then** // Check number of edges with max. consist. gain
12 $e_{switch} = e_1 \in E_{MaxCG}$
13 **else**
14 $e_{switch} = e_1 \in \{e \in E_{MaxCG} \| e_{sim} = min(\|\tau - e_{sim}\|)\}$ // Get best edge
15 Switch edge e_{switch} ($D \rightarrow ND$ or $ND \rightarrow D$) // Perform edge switch
16 **if** *c is split in 2 components* **then** // Check component split
17 $c \leftarrow$ component with more vertices
18 add component with less vertices to C
19 $RC \leftarrow$ new cluster with vertices in c // Create new cluster
20 **return** RC

21 **Function** `get_cg`(e, g): // Get consistency gain of edge e in graph g
22 $count \leftarrow$ number of consistent triangles in g
23 $count_{switched} \leftarrow$ number of consistent triangles in g with switched edge classification for e
24 **return** $count - count_{switched}$

the selected connected component or an edge switch would result in a negative consistency gain, then the elements of the selected component are added as a new cluster to the result set (line 19).

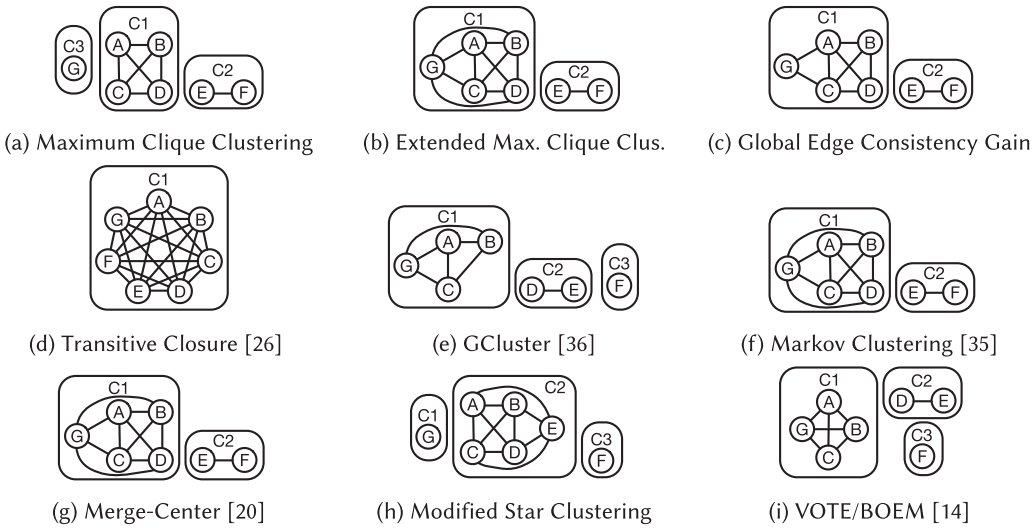
5 EXISTING CLUSTERING ALGORITHMS

In this section, we describe six clustering algorithms from related work. These algorithms were evaluated in References [19, 36] and are among the best clustering algorithms for duplicate detection.

Some of these clustering algorithms require a similarity score for each edge (weighted edges). The other clustering algorithms perform their clustering on the structure of the input graph (unweighted edges only). To illustrate the effects of the different clustering algorithms, we use the sample result of a pairwise comparison in Figure 7(a). Table 4 gives an overview of the clustering algorithms and shows which clustering algorithms require a similarity score and how many edge switches are needed for clustering the sample input graph in Figure 7(a). Note that in Figure 7(a) all missing edges mean a classification as non-duplicate. So adding a missing edge means changing the classification from non-duplicate to duplicate and removing an edge vice versa. The different clustering algorithm results for the sample input graph are shown in Figure 11, with Figures 11(a)–11(c) showing the results of the previously presented algorithms.

Table 4. Overview of the Nine Clustering Algorithms and the Number of Edge Switches for the Example in Figure 7(a)

Clustering Alg.	Needs sim.	# edge switches D→ND	# edge switches ND→D
Maximum Clique Clustering	no	3	0
Extended Max. Clique Cl.	no	1	2
Global Edge Consist. Gain	yes	1	0
Transitive Closure	no	0	11
GCluster	yes	4	1
Markov Clustering	yes	1	2
Merge Center Clustering	yes	1	2
Modified Star Clustering	no	3	4
VOTE/BOEM	yes	4	1



(a) Maximum Clique Clustering (b) Extended Max. Clique Clus. (c) Global Edge Consistency Gain

(d) Transitive Closure [26] (e) GCluster [36] (f) Markov Clustering [35]

(g) Merge-Center [20] (h) Modified Star Clustering (i) VOTE/BOEM [14]

Fig. 11. Clustering result of the different clustering algorithms for the sample graph in Figure 7(a).

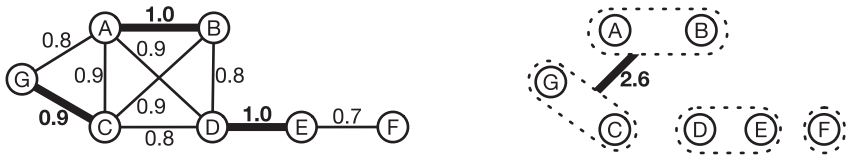
5.1 Transitive Closure

The transitive closure approach is based on the observation, that the relation "is-duplicate-of" is transitive [26]. Two vertices belong to the same cluster if a path exists. The transitive closure only switches edges from non-duplicate to duplicate, but not vice versa. So it increases the recall at cost of precision.

Note that there are many algorithms for calculating the transitive closure, e.g., from Warshall [41] or Warren [40], but most of them, including the aforementioned papers, are for directed graphs, which is more complex than for undirected graphs. Figure 11(d) shows the result of the transitive closure for our example. Overall, 11 edges are switched from non-duplicate to duplicate.

5.2 GCluster

The concept of GCluster [36] is creating *δ-cliques* to maximize the cohesion of the elements in the *δ-cliques*. In a *δ-clique* with v vertices, every vertex is connected to at least $\delta(v-1)$ vertices in the



(a) (b)

Fig. 12. GCluster maximum weight matchings, with $\delta = 0.6$. An edge implies that merging the vertices creates a new δ-*clique*. The thick edges show the result of the maximum weight matching.

δ-*clique*. Cohesion is a fitness measure for a subgraph and defined as the sum of weights of the edges in the subgraph.

GCluster first removes all edges from the input graph that have a similarity weight below a threshold and then calculates the maximum weight matching [16]. A maximum weight matching for a graph is a matching, in which the edge weight sum is maximal. The result of the weighted matching are connected components that are merged. If we have two merged components A and B, then the edge between A and B has as weight the sum of edge weights of vertices in A to vertices in B. The process of calculating the maximum edge weight and merging the connected components is repeated until no components can be merged. The maximum weight matching considers only edges between components, which is still a δ-*clique* after being merged.

Figure 12 shows the GCluster algorithm for our example in Figure 7(a) with a threshold $\theta = 0.5$ and $\delta = 0.6$. All edges are above the threshold and Figure 12(a) shows the maximum weight matching. Each pair $\langle A, B\rangle$, $\langle C, G\rangle$, and $\langle D, E\rangle$ is merged. In the second iteration (see Figure 12(b)), there is only one edge left that fulfills the requirement that merging the connected components would lead to to a δ-*clique*. The edge weight is the sum of edge weights between $\langle A, C\rangle$, $\langle A, G\rangle$, and $\langle C, B\rangle$. The result of GCluster for the example is shown in Figure 11(e).

In consultation with the authors of GCluster, our implementation uses an adapted version of the GCluster pseudo-code (Algorithm 1 in Reference [36]). We adjusted in line 10 the test whether the merged graph is a δ-*cliques* or not, analogous to the description in the paper.

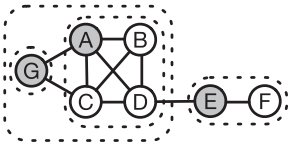
5.3 Markov Clustering

Markov Clustering (MCL) [35] is an unsupervised clustering algorithm that simulates random walks (or flows) in a graph by alternating an expansion and an inflation step on the associated Markov matrix. The underlying idea is that clusters are regions in a graph with many edges. In case a walk visits a dense cluster, it likely visits many vertices of that cluster before it finds a way out of the cluster. Thus, there are regions with a high flow in the graph and regions with a low flow. By applying simple algebraic operations on the associated Markov matrix, MCL strengthens areas in the graph with a high flow and weakens areas with a low flow. The expansion step calculates the normal matrix product, whereas the inflation step calculates the Hadamard power [23] followed by a scaling step. For our evaluation, we used the originally implementation.[3] Figure 11(f) shows the result of Markov clustering for our example.

5.4 Merge-Center Clustering

Merge-Center [20] is an extension of Center [21], where the idea is that every cluster has a center vertex and all other elements in that cluster are similar to this center vertex. As shown in Reference [19], Merge-Center outperforms Center.

[3] http://micans.org/mcl/index.html.



Edge	Sim
(A,B)	1.0
(A,C)	0.9
(A,D)	0.9
(B,C)	0.9
(C,G)	0.9
(D,E)	0.9
(A,G)	0.8
(B,D)	0.8
(C,D)	0.8
(E,F)	0.7

Fig. 13. Merge Center clustering with shaded center nodes.

Vertex	Degree
A	4
C	4
D	4
B	3
E	2
G	2
F	1

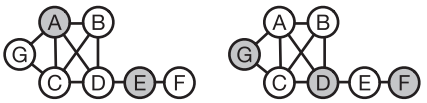


Fig. 14. Star Clustering for the sample in Figure 7(a). The table shows the degree for each vertex and the graphs show two possible solutions for star clustering with shaded star centers.

First, Merge-Center sorts all edges by their weight (similarity) in descending order and then sequentially scans these vertex pairs. The first time a vertex v_n occurs in the scan, it is assigned as center vertex of a new cluster. All subsequent vertices v_m that appear in pairs of the form $\langle v_n, v_m \rangle$ are assigned to the cluster of v_n.

The extension of Merge-Center compared to Center is that in case of processing a pair $\langle v_n, v_m \rangle$, with v_m being already an element of a different cluster than v_n, the clusters of v_n and v_m are merged. The merged cluster then has multiple center vertices. Due to the merge step, Merge-Center creates fewer clusters than Center.

Figure 13 shows the Merge-Center algorithm for the sample graph, with shaded center nodes. When processing the edge $\langle A, G \rangle$, the two clusters with center nodes A and G are merged. The result is shown in Figure 11(g).

5.5 Modified Star Clustering

Star Clustering [1] covers a similarity graph with star-shaped dense subgraphs. The star-shaped subgraphs consist of a single star center and satellite vertices. The similarity of each satellite to its star center is above a threshold. Applied to the duplicate detection process, this means that only edges classified as duplicates are considered.

Star Clustering first sorts all vertices by their degree in descending order. In the beginning, all vertices are unmarked and the algorithm iterates over the sorted list of vertices. If a vertex is unmarked, then it becomes the star center of a new cluster. All associated vertices become satellites in the cluster and are marked, so that they cannot become the star center of a new cluster. Note that Star Clustering creates overlapping clusters, if a vertex is associated to multiple star centers. This violates the goals for a clustering algorithm (see Section 2). Therefore, we are using a modified version of Star Clustering, in which a vertex is associated only to the first star center.

Thus, there are several possible solutions for star clustering. If several vertices have the same degree, then one is chosen as star center arbitrarily, e.g., the first one. Figure 14 shows two possible star clusterings for our sample graph, one with vertices A and E and one with vertices D, F, and G

as star centers. Figure 11(h) shows for the latter the created clusters of our modified star clustering approach.

5.6 Correlation Clustering

Correlation Clustering, introduced in Reference [2], tries to find a clustering in a complete graph, in which the edges are labeled either as + or −, depending on the classification of the vertices as being similar or dissimilar. The goal is to find a clustering that minimizes the number of + edges between clusters and the number of − edges within clusters. As shown in Reference [2], finding the best clustering is NP-hard, so there is a necessity for approximation algorithms.

Elsner and Schudy [14] compare several correlation clustering algorithms and recommend VOTE/BOEM for general problems. According to them, the input is a complete, undirected graph G with n nodes in which each edge has a probability p_{ij} whether nodes i and j belong to the same cluster. For duplicate detection, the probability corresponds to the similarity that we calculated for the pairwise classification. The goal is to find a clustering, defined as new graph G' with edges $x_{ij} = 1$ if nodes i and j belong to the same cluster, and otherwise $x_{ij} = 0$. Additionally, $x_{ii} = 1$ and $x_{ij} = x_{jk} = 1$ implies $x_{ij} = x_{ik}$.

The objective according to Reference [14] is to find a clustering that is as consistent as possible with regard to the given probabilities. Edges with a high probability should be within a cluster, but not crossing cluster boundaries. The opposite holds for edges with a low probability. The authors define w_{ij}^{+} as the cost of cutting an edge, with $w_{ij}^{+} = log(p_{ij})$, and w_{ij}^{-} as the cost of keeping an edge, with $w_{ij}^{-} = log(1 - p_{ij})$. Mathematically, the objective is

$$min \sum_{ij:i<j} x_{ij} w_{ij}^{-} + (1 - x_{ij}) w_{ij}^{+}. \tag{1}$$

VOTE/BOEM consists of two algorithms, VOTE and BOEM, that are executed sequentially. VOTE is a greedy algorithm that uses the *net weight* $w_{ij}^{\pm} = w_{ij}^{+} - w_{ij}^{-}$ to assign elements to an existing cluster or to create a new singleton cluster, as shown in Algorithm 3.

Elsner and Schudy [14] ran 100 random permutations and report the run with the best objective value. In our implementation, we run the algorithm for each connected component with up to 120 permutations. Thus. for all connected components with up to five vertices we run all possible permutations. Running the VOTE algorithm for each connected component instead on the whole graph reduces the complexity of the algorithm and is reasonable, because if two vertices are from different connected components, the net weight is always negative and these two vertices would never be added to the same cluster.

The clustering result of the VOTE algorithm is then the input for the *B*est *O*ne *E*lement *M*ove (BOEM) algorithm. This algorithm iteratively selects one element from the current clustering and either moves this element to another cluster or creates a new singleton cluster. In each iteration, we calculate for every element the change of the objective value for all possible moves, and finally execute the move with the highest optimization of the objective value. Note that for calculating the effect of an element move, we have to consider (i) the effect on the current cluster (element is removed), and (ii) the effect of the target cluster (element is added). The algorithm runs until there is no element move left that would improve the objective value. Figure 11(i) shows the result of VOTE/BOEM for the sample graph.

5.7 Complexity Analysis

We give a short overview of the time complexity for the presented algorithms. As the algorithms do not add vertices from different connected components to the same cluster, we can run each algorithm (except for Markov Clustering) per connected component, reducing complexity in

ALGORITHM 3: VOTE

Input: A complete, undirected graph G with n nodes; each edge in the graph has a probability p_{ij} whether nodes i and j belong to the same cluster.
Output: Clustering C

```
1  k ← 0                                                    // Number of clusters created so far
2  C[k++] ← {1}                                             // Create cluster for 1st vertex
3  for i = 2..n do                                          // Iterate over vertices
4      for c = 1..k do                                      // Iterate over clusters
5          Quality_c ← Σ_{j∈C[c]} w_ij^±                    // Get net weights sum
6      c* ← argmax_{1≤c≤k} Quality_c                        // Get best cluster
7      if Quality_c > 0 then
8          C[c*] ← C[c*] ∪ {i}                              // Add vertex to cluster
9      else
10         C[k++] ← {i}                                     // Form a new cluster
11 return C
```

practice with the worst case that all nodes are within a single component. For finding the connected components, we use a depth-first search, which has a complexity of $O(V + E)$, where V is the number of vertices, and E is the number of edges in a connected component.

For Transitive Closure, all elements in a connected component belong to the same cluster, so the complexity is $O(1)$. GCluster has a complexity of $O(V^{3.5})$ [36]. For Markov Clustering, a naive implementation has a complexity of $O(V^3)$, but by applying a pruning scheme, it can be reduced to $O(Vk^2)$, with k as a pruning constant that reduces the computation of a column to the k largest entries [35]. Merge-Center Clustering first sorts all edges by their similarity and then scans all edges, resulting in a complexity of $O(E \log(E) + E)$. Modified Star Clustering first calculates the degree of the vertices, then sorts them by their degree, and finally scans the vertices, so the complexity is $O(V + V \log(V) + V)$. The VOTE algorithm iterates over all vertices and calculates the net weight for all other processed vertices. In the end, the net weight is calculated for all pairs of vertices, so the complexity is $O(\frac{V^2-V}{2})$, multiplied by the number of permutations. As mentioned in Reference [14], BOEM can be implemented with a complexity of $O(V^2)$ for preprocessing, and $O(V)$ for each move, with at most $V - 1$ moves [17].

The complexity of MCC is dominated by the complexity for finding a maximum clique. For this, the Bron-Kerbosh algorithm has a complexity of $O(3^{V/3})$ [5]. As we do not have to find all maximal cliques, the complexity can be improved to $O(2^{V/3})$ [34] or $O(2^{0.276V})$ [31]. For EMCC, we additionally have to consider the complexity for the extension step, which is $O(V)$.

For GECG, as mentioned in Section 4, we have $V * (V - 1) * (V - 2)/6$ triangles in each component. Initially, we calculate in constant time for each triangle whether it is consistent. To choose the best edge, we calculate for all $V * (V - 1)/2$ edges the effect of an edge switch for $V - 2$ triangles, and then choose the best edge to be switched. In the following iterations, under the premise that we saved preliminary results, we recalculate only the consistency of $V - 2$ triangles, and the effect of an edge switch for $(V - 2) * 2 + 1$ edges (two edges per triangle affected by the previous edge switch plus the switched edge itself). As the number of inconsistent triangles is monotonically decreasing due to the condition that the consistency gain has to be positive (line 11 of Algorithm 2), the number of iterations depends on the number of triangles ($O(V^3)$), therefore the overall complexity is in $O(V^5)$.

Table 5. Complexity Comparison for EMCC and GECG

V	EMCC $O(3^{V/3})$	GECG $O(V^5)$
53	268,622,364	418,195,493
54	387.420.489	459.165.024
55	558.757.034	503.284.375
56	805.867.092	550.731.776

If we compare the complexity of the new clustering approaches MCC/EMCC and GECG, then for all three the runtime depends on the size of the connected components that result from the pairwise comparison. For smaller connected components with $V < 55$, MCC and EMCC are expected to run faster than GECG, and vice versa for larger connected components with $V \geq 55$ GECG is expected to be faster. This is shown in Table 5 and also later in our experimental evaluation.

6 EVALUATION

In this section, we evaluate all clustering algorithms presented in the previous sections, using various datasets. We employ different algorithms for the pairwise comparison. Next to an exhaustive comparison, we also evaluate the effects of Blocking and the Sorted Neighborhood method on the clustering result [7, 25]. Both these methods compare only record pairs with a higher chance of being duplicates and thus reduce the pairwise comparison effort with the restriction that some duplicates might be missed.

6.1 Baseline Clustering Algorithms

For our evaluation, we additionally use two baseline algorithms that help to measure the performance of the clustering algorithms.

NoClustering: No clustering means that no edge is switched and the clustering result is identical to the result of the pairwise comparison. Thus, we do not have cliques as described in Section 2, and the result might be inconsistent regarding the "is-duplicate-of" relation, e.g., $\langle A, B\rangle$ and $\langle B, C\rangle$ are duplicates, but $\langle A, C\rangle$ is not. The result of this approach can be used to evaluate how much a clustering algorithm can improve the result of the pairwise comparison. Due to possible inconsistencies in the result, this approach should not be used in real-life scenarios.

Gold Standard Clustering: We use the gold standard to decide for each component, which vertices belong to the same cluster. Thus, the precision of gold standard clustering is always 100%. Note that the recall is not always 100%, because gold standard clustering only considers duplicates that were placed in the same component by the pairwise classification. Gold standard clustering is not suitable in real-world scenarios, because the gold standard is generally unknown. But for our experiments, it is the upper bound for what can be reached without connecting different components.

6.2 Datasets

For our evaluation, we use various synthetic and real-world datasets. For each, we describe its content, our similarity measure, and other used parameters. Table 6 gives an overview of all datasets and Table 7 describes the connected components that result from the pairwise comparison.

The **Cora** citation matching dataset[4] comprises 1,879 references of research papers and is often used in the duplicate detection research [4, 10]. The definition of a *Cora* gold standard is described in Reference [11]. For the classification as duplicate or non-duplicate, we use a similarity measure

[4] http://people.cs.umass.edu/~mccallum/data.html.

Table 6. Overview of Datasets, Showing the Number of Records and Clusters, the Percentage of Singleton Clusters, and for Non-singleton Clusters the Cluster sizes

Dataset	Records	Cluster	Singleton	Non-singleton Clust. Sizes		
	#	#	%	Average	Median	Maxim.
Cora	1,879	182	35.16 %	15.38	6.0	238
CD	9,763	9,508	97.68 %	2.15	2.0	6
Febrl Small	11,000	10,000	94.99 %	3.00	3.0	4
Febrl Large	20,000	10,000	79.73 %	5.93	6.0	10
NCVoter	8,261,838	8,110,137	98.17 %	2.02	2.0	6
Birth records	17,611	5,244	41.15 %	5.01	5.0	16
Stringer	4,163	638	14.97 %	8.32	8.1	20.66

For *Stringer*, the table shows average values.

Table 7. Overview of Connected Components Without Singletons that Result from the Pairwise Comparison, Showing the Different Sizes Per Dataset and Pair-selection Algorithm (Average Values for *Stringer*)

Dataset	Exhaustive				Blocking				Sorted Neighborhood			
	Count	Avg	Median	Max	Count	Avg	Median	Max	Count	Avg	Median	Max
Cora	104	17.47	5	240	107	16.98	5	240	108	16.80	5	239
CD	204	2.19	2	6	199	2.16	2	6	200	2.16	2	6
Febrl Small	505	2.95	3	4	503	2.95	3	4	497	2.96	3	4
Febrl Large	2,024	6.08	5	826	2,061	5.90	5	334	2,066	5.68	5	77
NCVoter	—	—	—	—	179,059	2.04	2	9	168,785	2.04	2	9
Birth records	—	—	—	—	1,978	7.67	3	6,386	—	—	—	—
Stringer	398	20	9	614	—	—	—	—	—	—	—	—

that calculates the average Jaccard coefficient [26] of attributes *Title* and *Author* using bigrams. Additional rules set the similarity of a record pair to 0.0 when certain conditions are met. These rules are (1) the year attribute has different values, (2) one reference is a technical report and the other is not, (3) the Levenshtein edit distance of attribute *Pages* is greater than 2, and (4) one reference is a journal, but the other one is a book.

We use three different blocking keys for the Blocking experiments. These are the first 2 characters of attributes *ReferenceID*, *Title*, and *Author*, respectively. For the Sorted Neighborhood Method the window size is 20 and the used sorting keys are ⟨*Refer.ID, Title, Author*⟩, ⟨*Title, Author, Refer.ID*⟩, and ⟨*Author, Title, Refer.ID*⟩.

The **CD** dataset[5] is a randomly selected extract from freeDB.org. It contains information for 9,763 CDs, including artist, title, and songs. The dataset has been used in several papers [7, 22]. Our similarity function calculates the average Levenshtein similarity of the three attributes *Artist*, *Title*, and *Track01* but also considers *null* values and string-containment. For Blocking, we use again three blocking criteria, which are the first two characters of attributes *Artist*, *Title*, and *Track01*. The Sorted Neighborhood Method also uses a window size of 20 and the three sorting keys ⟨*Artist, Title, Track01*⟩, ⟨*Title, Artist, Track01*⟩, and ⟨*Track01, Artist, Title*⟩.

[5] http://www.hpi.uni-potsdam.de/naumann/projekte/dude_duplicate_detection.html.

The **Febrl** dataset generator [6] was used to create two artificial datasets. Both were created with 10,000 records of unique entities. The smaller dataset (**Febrl small**) contains an additional 1,000 duplicate records with up to three duplicates per entity. The larger dataset (**Febrl large**) contains 10,000 additional duplicate records with up to nine duplicates per cluster and additionally more modifications in the duplicates than those in the smaller one.

The similarity function aggregates the Jaro-Winkler similarities of attributes *Firstname*, *Lastname*, *Address*, *Suburb*, and *State*. Blocking uses three blocking criteria *Firstname*, *Lastname*, and *Postcode* (first two characters of each attribute), and the Sorted Neighborhood Method uses three sorting keys ⟨*Firstname, Lastname*⟩, ⟨*Lastname, Firstname*⟩, and ⟨*Postcode, Address*⟩ with window size 20.

The **NCVoter** dataset[6] contains about 8.2M records with personal details of individuals, such as name, address, and age. For some voter there exists more than one record, for instance, because their personal details have changed over time, or misspellings were corrected. Ramadan et al. have extensively deduplicated this dataset, and we use their result as gold standard [29].

Due to the high number of records, it is not feasible to perform an exhaustive pair-wise comparison. Thus, we evaluate only the clustering algorithms based on Blocking and the Sorted Neighborhood Method. Blocking considers the following two blocking criteria: the concatenation of the first two letters of *Firstname* and *Lastname* and the concatenation of the first four letters of *City* and *Street address*. The Sorted Neighborhood Method uses a window size of 20 and ⟨*Lastname, Firstname, Middle name*⟩, and ⟨*Firstname, Lastname, Middle name*⟩ as sorting keys.

The historical **Birth records** dataset consists of 17,614 birth records from the Isle of Skye in Scotland, spanning the years 1861 to 1901, where the linkage task is to group all birth records of babies with the same parents, i.e., create one cluster per family. For each birth record, name and address details of the baby and its parents, as well as marriage date and place, and occupation information of the parents were used in a pair-wise comparison step. A locality-sensitive hashing-based (LSH) blocking approach on these string attributes (converted into character bi-grams) was applied. The Jaro-Winkler approximate string comparison function was used to calculate the similarities between string values and an approximate numerical similarity was calculated between year values [8]. Due to the highly skewed nature of the names and addresses in this dataset, where the majority of people had one of only a very few common names and lived in a small number of villages [30], the basic pair-wise linkage did not result in high linkage quality (too many false matches) [9], and clustering is required to identify the correct groupings of birth records. Manually prepared ground truth, based on extensive semi-automatic linkages done by domain experts [30], was available, which allowed us to calculate the quality of the final clustering results.

The **Stringer** data collection[7] comprises 29 datasets that were created with an enhanced version of the UIS database generator and that were also used in the experimental evaluation in Reference [19]. We use *Stringer* only for an evaluation of clustering algorithms based on an exhaustive pair-wise comparison, as it would be too time-consuming to find good blocking criteria and sorting keys for each of the 29 datasets. The similarity measure corresponds to the implementation in Reference [19]. We use a weighted Jaccard similarity based on bigrams.

6.3 Evaluation Approach and Results

We used the DuDe toolkit [11] (Java-based) for our evaluation and ran all experiments on a server with two 6-core 64-bit Intel Xeon 2.93 GHz CPUs, 96 GBytes of memory and running Ubuntu

[6] ftp://www.app.sboe.state.nc.us.
[7] http://dblab.cs.toronto.edu/project/stringer/clustering/.

Table 8. EMCC: Best Value Ranges for τ

Dataset	Exhaustive Comparison	Blocking	Sorted Neighborhood
Cora	0.32 - 0.38	0.07	0.04
CD	0.51 - 1.00	0.51 - 1.00	0.51 - 1.00
Febrl Small	0.00 - 0.33	0.00 - 0.33	0.00 - 0.33
Febrl Large	0.21 - 0.22	0.15 - 0.16	0.11
NCVoter	—	0.41 - 0.50	—
Birth records	—	0.40 & 0.48 - 0.50	—
Stringer	0.50	—	—

17.04. For the clustering algorithms, we use the following configurations: GCluster with $\theta = 0.5$ and $\delta = 0.6$, and for EMCC, we evaluated the best values for τ, as shown in Table 8.

We use four different measures to evaluate the performance of the clustering algorithms. Next to the most prominent measures *precision* (fraction of correctly detected duplicates over all detected duplicates), *recall* (fraction of detected duplicate pairs over the overall number of existing duplicate pairs) and *f-measure* (harmonic mean of precision and recall) [26], we also use the *generalized merged distance (GMD)* [24]. GMD is defined as the minimal number of merge (m) and split (s) operations to transform the clustering result to the real-world classification. GMD can be configured in different ways, and we use the costs $f_m = 1$ and $f_s = 1$ in our evaluation, so that all clusters are equally important, independent of their size. The first three measures are pair-wise measures, whereas GMD considers the quality of the clusters.

We acknowledge recent work that has identified some issues when the F-measure is used to compare deduplication methods [18]. The harmonic mean calculation of the F-measure can be converted into a weighted arithmetic mean of precision and recall, where however different weights are assigned to precision and recall depending upon the number of classified duplicates. This can occur, for example, when different similarity thresholds are used when deduplication methods are compared. In our evaluation, however, as we discuss next, we do not vary such similarity thresholds for the same dataset, but rather we identify the best threshold for each dataset based on the exhaustive pairwise comparison. Therefore, our use of the F-measure is valid. Furthermore, we present precision and recall results as well to provide the full details of the obtained deduplication quality.

The first evaluation step is finding a suitable classification threshold for each dataset to classify the record pairs as duplicate or non-duplicate. To give no clustering approach an advantage, we take the f-measure result of the exhaustive pairwise comparison without any clustering as benchmark. If for one dataset multiple thresholds lead to the same f-measure value, then we take the threshold with the best precision value, and if there are still multiple possible thresholds, we take the highest one. Table 9 gives an overview of the datasets and their best threshold that we use in our evaluation. The table additionally shows precision and recall values. Due to the high number of records for *NCVoter*, here we use the Sorted Neighborhood results instead.

In the second step, we use the previously evaluated thresholds to run the clustering experiments. For *Stringer*, we have aggregated the results of the individual datasets and show only the average results in Table 10(a). This table shows for *NoClustering* the absolute values for f-measure, precision, and recall, and for all other clustering algorithms the difference in comparison to *NoClustering*. The highest improvements for each measure are highlighted, e.g., *GECG* has the best f-measure value, which is 2.98% higher than the f-measure value for *NoClustering*. The column for the GMD shows absolute values, not the improvements. Note that for *NoClustering*, we cannot

Table 9. Best Threshold and its Result Per Dataset for a Exhaustive Pairwise Comparison Without Clustering

Dataset	Best Th.	F-Meas.	Precision	Recall
Cora	0.64	97.66 %	98.05 %	97.28 %
CD	0.81	88.32 %	90.81 %	85.95 %
Febrl Small	0.91	97.07 %	99.24 %	95.00 %
Febrl Large	0.87	90.75 %	97.37 %	84.97 %
NCVoter	0.56	82.09 %	76.40 %	88.69 %
Birth records	0.78	67.18 %	66.40 %	67.98 %
ST. AB	0.64	99.72 %	99.80 %	99.63 %
ST. DBLPH1	0.18	87.77 %	91.59 %	84.25 %
ST. DBLPH2	0.23	86.53 %	88.76 %	84.41 %
ST. DBLPL1	0.36	99.50 %	99.75 %	99.26 %
ST. DBLPL2	0.40	99.63 %	99.61 %	99.64 %
ST. DBLPM1	0.26	89.65 %	91.14 %	88.20 %
ST. DBLPM2	0.32	94.53 %	97.77 %	91.51 %
ST. DBLPM3	0.29	99.18 %	99.03 %	99.34 %
ST. DBLPM4	0.31	99.01 %	98.69 %	99.33 %
ST. EDH	0.20	53.59 %	52.90 %	54.29 %
ST. EDL	0.35	87.98 %	89.77 %	86.27 %
ST. EDM	0.25	68.95 %	70.80 %	67.20 %

Dataset	Best Th.	F-Meas.	Precision	Recall
ST. H1	0.22	54.81 %	54.55 %	55.06 %
ST. H2	0.31	59.28 %	57.36 %	61.34 %
ST. L1	0.47	93.80 %	96.49 %	91.25 %
ST. L2	0.55	96.24 %	99.04 %	93.60 %
ST. M1	0.42	72.71 %	86.68 %	62.62 %
ST. M2	0.56	91.06 %	99.21 %	84.14 %
ST. M3	0.36	88.85 %	89.22 %	88.49 %
ST. M4	0.41	90.58 %	90.89 %	90.27 %
ST. TS	0.89	100.00 %	100.00 %	100.00 %
ST. ZH1	0.33	38.48 %	36.96 %	40.13 %
ST. ZH2	0.43	54.77 %	65.85 %	46.88 %
ST. ZL1	0.50	92.48 %	92.65 %	92.30 %
ST. ZL2	0.53	95.13 %	93.93 %	96.36 %
ST. ZM1	0.60	71.75 %	97.80 %	56.66 %
ST. ZM2	0.60	94.00 %	98.50 %	89.90 %
ST. ZM3	0.43	84.21 %	84.71 %	83.71 %
ST. ZM4	0.47	87.68 %	89.40 %	86.03 %

calculate a GMD, as we would need consistent clusters. For f-measure, precision, and recall, higher values are better, whereas the GMD should be as low as possible. The results of the *Birth records* and *NCVoter* datasets are shown in Tables 10(b) and 10(c), whereas Table 11 shows the results of datasets *Cora* (Table 11(a)) and *CD* (Table 11(b)), and Table 12 shows the results for both *Febrl* datasets, each with all pair selection algorithms.

For algorithm *GECG*, four *Stringer* and the *Birth records* dataset did not finish within 60 hours due to very large connected components (>3,500 vertices). Thus, for *GECG* and *Stringer*, we report only the average results of all other *Stringer* datasets, and for *Birth records* no values. We briefly report on runtimes separately at the end of the section.

For all datasets, we observe that our classifiers alone already lead to good results (see NoClustering). The possible improvements of a perfect clustering algorithm that improves the quality only within a component, but does not assign records of different components to the same cluster, is very limited (see GoldStandard clustering). However, a clustering algorithm can also worsen the results of a pairwise comparison.

We can distinguish between datasets with small cluster sizes (e.g., *CD*, *Febrl small*, *NCVoter*) and datasets with large cluster sizes (e.g., *Cora*, *Febrl large*, *Birth records*). For the former datasets, the clustering algorithms achieve the same or similar results and there is no single clustering algorithm that outperforms the others.

For the latter datasets, we observe that the Transitive Closure, as expected, leads to the best recall values; however, it also results in very low precision and thus low f-measure values. Merge-Center Clustering and Modified Star Clustering (with the exception of *Febrl large*) also tend to decrease the quality of the results. The best performing clustering algorithms are Markov clustering and EMCC. The main difference between the results of these two algorithms is that Markov Clustering leads to higher recall values, whereas EMCC leads to a higher precision.

Table 10. Experimental Evaluation for Datasets Stringer, Birth Records, and NCVoter with Different Pair-selection Strategies

(a) Results Stringer dataset

Stringer	Exhaustive Comparison			
Cluster Alg.	F-Measure	Precision	Recall	GMD
No Clustering	83.86 %	86.65 %	81.80 %	—
Gold Standard Clust.	+10.00 %	+13.35 %	+7.96 %	157.59
MCC	−6.86 %	+2.76 %	−12.32 %	617.79
EMCC	+0.10 %	+1.13 %	−0.67 %	414.76
GECG	+2.98 %	+3.24 %	+2.84 %	275.56
Transitive Closure	−19.23 %	−25.16 %	+7.96 %	272.79
GCluster	−1.70 %	+8.50 %	−6.55 %	432.03
Markov Clustering	+2.40 %	−1.22 %	+7.22 %	239.55
Merge-Center Clust.	−18.39 %	−24.21 %	+6.84 %	288.62
Mod. Star Clustering	−2.86 %	−6.53 %	+1.47 %	414.24
VOTE/BOEM	−14.59 %	+10.50 %	−21.18 %	1,010.31

(b) Results Birth records dataset

Birth Records	LSH-based Blocking			
Cluster Alg.	F-Measure	Precision	Recall	GMD
No Clustering	67.18 %	66.40 %	67.98 %	—
Gold Standard Clust.	+24.05 %	+33.60 %	+15.89 %	1,273
MCC	+1.53 %	+22.29 %	−11.90 %	3,808
EMCC	+5.45 %	+14.68 %	−2.21 %	3,293
GECG	—	—	—	—
Transitive Closure	−66.84 %	−66.23 %	+15.89 %	3,366
GCluster	+1.50 %	+24.84 %	−12.91 %	3,713
Markov Clustering	−1.43 %	−9.11 %	+9.17 %	2,899
Merge-Center Clust.	−66.76 %	−66.19 %	+13.21 %	3,449
Mod. Star Clustering	−16.68 %	−26.06 %	−0.48 %	3,912
VOTE/BOEM	+2.13 %	+24.01 %	−11.79 %	3,647

(c) Results NCVoter dataset

NCVoter	Blocking				Sorted Neighborhood			
Clustering Alg.	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD
No Clustering	83.37 %	75.35 %	93.31 %	—	82.09 %	76.40 %	88.69 %	—
Gold Standard Clust.	+13.36 %	+24.65 %	+0.35 %	9,656	+12.05 %	+23.60 %	+0.25 %	16,777
MCC	−0.13 %	+0.21 %	−0.67 %	54,331	−0.09 %	+0.19 %	−0.45 %	57,102
EMCC	−0.11 %	−0.36 %	+0.28 %	54,057	−0.09 %	+0.19 %	−0.45 %	57,089
GECG	−0.09 %	+0.22 %	−0.55 %	54,086	−0.07 %	+0.17 %	−0.38 %	56,976
Transitive Closure	−0.20 %	−0.55 %	+0.35 %	54,099	−0.14 %	−0.42 %	+0.25 %	56,987
GCluster	−0.05 %	+0.27 %	−0.55 %	54,056	−0.05 %	+0.20 %	−0.38 %	56,960
Markov Clustering	−0.17 %	−0.51 %	+0.34 %	54,089	−0.13 &	−0.39 %	+0.23 %	56,983
Merge Center Clust.	−0.11 %	−0.19 %	+0.01 %	54,019	−0.10 %	−0.15 %	−0.02 %	56,954
Mod. Star Clustering	−0.17 %	−0.49 %	+0.33 %	54,086	−0.12 %	−0.37 %	+0.23 %	56,975
VOTE/BOEM	−0.05 %	+0.30 %	−0.57 %	54,045	−0.04 %	+0.24 %	−0.39 %	56,937

For the difficult *Cora* dataset, EMCC is the only algorithm that shows a slight increase for f-measure, while Markov Clustering shows only a slight decrease. For *Febrl large*, they both show the best results and for *Birth records* EMCC has the highest f-measure value. GCluster shows especially good results for precision, but yields the lowest recall values. Thus, GCluster is especially useful in scenarios where high precision values are required.

The extension of MCC to EMCC shows a positive effect, especially on the recall values. Due to possible false pair-wise classifications, not every component is a clique. MCC splits these components and creates smaller clusters, wheras EMCC corrects possible false classifications, which results in larger clusters and thus higher recall values. For *Febrl large*, we also observe a higher precision.

The effect of the pair selection algorithm on the ranking of clustering algorithms is very low. If a clustering algorithm shows the best result for an exhaustive comparison, then it is also one of the best algorithms for this dataset with Blocking or the Sorted Neighborhood Method. In general, the GMD values confirm our observations and interpretations for all datasets and clustering methods.

Table 11. Experimental Evaluation for Datasets Cora and CD with Different Pair-selection Strategies

(a) Results Cora dataset

Cora	Exhaustive Comparison				Blocking				Sorted Neighborhood			
Clustering Alg.	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD
No Clustering	97.66 %	98.05 %	97.28 %	—	94.88 %	98.87 %	91.20 %	—	55.20 %	98.68 %	38.32 %	—
Gold Standard Clust.	+2.10 %	+1.95 %	+2.23 %	12	+4.88 %	+1.13 %	+8.31 %	12	+44.56 %	+1.32 %	+61.19 %	12
MCC	−1.48 %	+1.29 %	−4.07 %	40	−4.98 %	+0.61 %	−9.19 %	59	−22.16 %	+0.72 %	−18.51 %	115
EMCC	+0.21 %	+0.21 %	+0.21 %	32	+3.99 %	−0.52 %	+8,19 %	29	+21.74 %	−0.51 %	+24,94 %	40
GECG	−0.77 %	−1.34 %	−0.21 %	33	+0.43 %	+0.32 %	+0.53 %	44	−5.45 %	+0.57 %	−5.13 %	62
Transitive Closure	−9.65 %	−19.15 %	+2.23 %	40	−4.88 %	−16.72 %	+8.31 %	37	+36.06 %	−14.42 %	+61.19 %	33
GCluster	−3.86 %	+0.95 %	−8.16 %	53	−26.55 %	−0.05 %	−38.98 %	77	−24.77 %	−0.08 %	−20.33 %	124
Markov Clustering	−0.32 %	−2.78 %	+2.23 %	28	+3.63 %	−1.27 %	+8.24 %	28	+18.63 %	−0.60 %	+20.87 %	28
Merge-Center Clust.	−9.65 %	−19.15 %	+2.23 %	40	−4.04 %	−15.30 %	+8.31 %	35	+36.85 %	−13.05 %	+61.19 %	31
Star Clustering	−4.15 %	−9.40 %	+1.65 %	44	−1.69 %	−6.32 %	+2.64 %	50	+2.19 %	−4.01 %	+2.86 %	62
VOTE/BOEM	−8.95 %	+0.53 %	−16.65 %	51	−10.06 %	−0.09 %	−16.89 %	65	−10.80 %	+0.80 %	−9.74 %	100

(b) Results CD dataset

CD	Exhaustive Comparison				Blocking				Sorted Neighborhood			
Clustering Alg.	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD
No Clustering	88.32 %	90.81 %	85.95 %	—	90.18 %	94.83 %	85.95 %	—	90.02 %	94.49 %	85.95 %	—
Gold Standard Clust.	+4.32 %	+9.19 %	+0.34 %	38	+2.46 %	+5.17 %	+0.34 %	38	+2.62 %	+5.51 %	+0.34 %	38
MCC	+1.19 %	+2.96 %	−0.33 %	56	+0.12 %	+0.69 %	−0.33 %	51	+0.12 %	+0.68 %	−0.33 %	52
EMCC	+1.19 %	+2.96 %	−0.33 %	56	+0.12 %	+0.69 %	−0,33 %	51	+0.12 %	+0.68 %	−0.33 %	52
GECG	+1.19 %	+2.96 %	−0.33 %	56	+0.12 %	+0.69 %	−0.33 %	51	+0.12 %	+0.68 %	−0.33 %	52
Transitive Closure	−1.89 %	−4.23 %	+0.34 %	64	−0.13 %	−0.67 %	+0.34 %	52	−0.12 %	−0.67 %	+0.34 %	53
GCluster	+1.19 %	+2.96 %	−0.33 %	56	+0.12 %	+0.69 %	−0.33 %	51	+0.12 %	+0.68 %	−0.33 %	52
Markov Clustering	−1.89 %	−4.23 %	+0.34 %	64	−0.13 %	−0.67 %	+0.34 %	52	−0.12 %	−0.67 %	+0.34 %	53
Merge-Center Clust.	+0.88 %	+2.28 %	−0.33 %	57	−0.20 %	−0.01 %	−0.33 %	52	−0.19 %	−0.02 %	−0.33 %	53
Mod. Star Clustering	−1.89 %	−4.23 %	+0.34 %	64	−0.13 %	−0.67 %	+0.34 %	52	−0.12 %	−0.67 %	+0.34 %	53
VOTE/BOEM	−1.60 %	−3.65 %	+0.34 %	63	−0.13 %	−0.67 %	+0.34 %	52	−0.12 %	−0.67 %	+0.34 %	53

Runtime results. To ensure a comparison that is as fair as reasonably possible, we re-implemented all clustering approaches in Java. Only two exceptions were made: GCluster includes a maximum weight-matching step, for which we used an existing Python library NetworkX,[8] and Markov-Clustering, for which we used the published C implementation of the author.[9] Table 13 reports exemplary runtimes for each algorithm for three selected datasets, one with large clusters (*Cora*) and two with small clusters (*CD*, *Febrl small*) and the exhaustive pair-selection strategy.

The reported runtimes reflect only the time needed for the clustering step itself, but not time spent on pair classification. Each experiment was run five times, and we report average runtimes.

We observe that runtime behavior can depend on cluster sizes. The *Cora* dataset contains very large clusters, while the other two datasets contain only many small clusters. Some algorithms cope well with this difference, while others are significantly slower in the former case. This is, in particular, true for five algorithms: Large clusters lead to high complexities for calculating maximum cliques (MCC, EMCC), triangles (GECG), matchings (GCluster), or net weight (VOTE/BOEM).

[8] https://networkx.github.io.
[9] https://micans.org/mcl.

Table 12. Experimental Evaluation for Febrl Datasets with Different Pair-selection Strategies

(a) Results Febrl Small dataset

Febrl Small	Exhaustive Comparison				Blocking				Sorted Neighborhood			
Clustering Alg.	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD
No Clustering	97.07 %	99.24 %	95.00 %	—	97.10 %	99.37 %	94.94 %	—	97.25 %	99.75 %	94.88 %	—
Gold Standard Clust.	+1.00 %	+0.76 %	+1.20 %	29	+0.97 %	+0.63 %	+1.26 %	29	+0.82 %	+0.25 %	+1.32 %	29
MCC	−0.92 %	−0.01 %	−1.75 %	60	−0.99 %	−0.01 %	−1.87 %	59	−1.06 %	−0.01 %	−1.99 %	54
EMCC	+0.63 %	+0.01 %	+1.20 %	41	+0.66 %	+0.01 %	+1.26 %	39	+0.69 %	+0.00 %	+1.32 %	33
GECG	+0.04 %	+0.01 %	+0.06 %	50	+0.07 %	+0.00 %	+0.12 %	48	+0.10 %	+0.00 %	+0.18 %	42
Transitive Closure	+0.63 %	+0.01 %	+1.20 %	41	+0.66 %	+0.01 %	+1.26 %	39	+0.69 %	+0.00 %	+1.32 %	33
GCluster	+0.00 %	+0.00 %	+0.00 %	50	+0.03 %	+0.00 %	+0.06 %	48	+0.06 %	+0.00 %	+0.12 %	42
Markov Clustering	+0.63 %	+0.01 %	+1.20 %	41	+0.66 %	+0.01 %	+1.26 %	39	+0.69 %	+0.00 %	+1.32 %	33
Merge-Center Clust.	+0.45 %	+0.01 %	+0.84 %	44	+0.48 %	+0.01 %	+0.90 %	42	+0.51 %	+0.00 %	+0.96 %	36
Mod. Star Clustering	+0.63 %	+0.01 %	+1.20 %	41	+0.66 %	+0.01 %	+1.26 %	39	+0.69 %	+0.00 %	+1.32 %	33
VOTE/BOEM	−0.92 %	−0.01 %	−1.75 %	60	−0.99 %	−0.01 %	−1.87 %	59	−1.06 %	−0.01 %	−1.99 %	54

(b) Results Febrl Large dataset

Febrl Large	Exhaustive Comparison				Blocking				Sorted Neighborhood			
Clustering Alg.	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD	F-Measure	Precision	Recall	GMD
No Clustering	90.75 %	97.37 %	84.97 %	—	90.17 %	97.94 %	83.55 %	—	88.31 %	99.15 %	79.62 %	—
Gold Standard Clust.	+6.18 %	+2.63 %	+9.07 %	396	+6.45 %	+2.06 %	+9.91 %	428	+7.21 %	+0.85 %	+11.81 %	545
MCC	−7.81 %	+1.48 %	−13.53 %	1,879	−8.83 %	+1.02 %	−14.51 %	1,938	−10.53 %	+0.34 %	−15.77 %	2,089
EMCC	+3.38 %	−0.20 %	+6.31 %	941	+3.78 %	−1.20 %	+7.77 %	925	+5.70 %	−1.35 %	+10.88 %	789
GECG	+1.61 %	+1.57 %	+1.63 %	1,089	+1.66 %	+1.15 %	+2.01 %	1,122	+1.34 %	+0.47 %	+1.86 %	1,218
Transitive Closure	−74.09 %	−88.23 %	+9.07 %	1,066	−38.37 %	−62.11 %	+9.91 %	959	+1.29 %	−11.31 %	+11.81 %	759
GCluster	−2.71 %	+1.34 %	−5.52 %	1,304	−2.79 %	+0.94 %	−5.27 %	1,295	−4.19 %	+0.30 %	−6.73 %	1,340
Markov Clustering	+3.86 %	−1.81 %	+8.70 %	883	+4.64 %	−1.36 %	+9.55 %	831	+6.34 %	−0.63 %	+11.44 %	736
Merge-Center Clust.	−59.18 %	−78.34 %	+7.51 %	991	−16.06 %	−35.80 %	+8.24 %	927	+3.68 %	−4.39 %	+9.76 %	832
Mod. Star Clustering	+0.76 %	−4.89 %	+5.60 %	1,107	+1.99 %	−3.77 %	+6.69 %	1,031	+4.59 %	−1.58 %	+9.03 %	884
VOTE/BOEM	−8.09 %	+1.54 %	−13.97 %	1,820	−8.95 %	+1.05 %	−14.70 %	1,905	−10.66 %	+0.36 %	−15.95 %	2,045

The impact of the clustering step on the overall runtime depends mainly on the size of the connected components created in the pairwise-comparison step. Table 14 shows the runtime for the pairwise comparison. It mainly depends on the dataset size but also on the complexity of the classifier. For Blocking and the Sorted Neighborhood Method, the runtimes are much smaller, as expected. In case of large connected component sizes, e.g., for Cora, we can see that the selection of the clustering algorithm has a high impact on the overall runtime. Vice versa, for small connected component sizes the selection of the clustering algorithm has only a small impact on the overall runtime, which is mainly dominated by the runtime of the pair-selection algorithm.

Evaluation of the effect of different classifiers on the clustering result. For the previous experiments, we used high-quality classifiers, which led to good results for a pairwise comparison. Misclassifications can, but do not necessarily, have a negative impact on the overall result. Classifiers can be very restrictive, which leads to smaller connected components with a high precision, but a low recall value. Vice versa, classifiers can classify too many record pairs as duplicate and therefore increase the size of the connected components, which possibly leads to a higher recall value, but a lower precision. We now experimentally evaluate the effect of misclassifications and

Table 13. Clustering Runtimes (in ms) for an Exhaustive Comparison of Selected Datasets

Cluster Alg.	Cora	CD	Febrl small
MCC	767,559	86	136
EMCC	785,137	91	141
GECG	11,131	86	131
Transitive Closure	781	61	98
GCluster	155,708	98,034	354.923
Markov Clustering	598	9	33
Merge Center Clust.	399	21	35
Mod. Star Clustering	792	68	103
VOTE/BOEM	8,196	74	112

Table 14. Pair-creation Runtimes (in ms)

Dataset	Exhaustive	Blocking	SNM
Cora	176,565	28,148	10,354
CD	280,241	18,334	5,233
Febrl Small	414,740	30,178	5,858
Febrl Large	1,240,957	100,900	9,380
NCVoter	—	23,523,221	2,706,307
Stringer	100,471	—	—

The table shows the average values of five runs.

Table 15. Three Classifiers for the Cora Dataset

Classifier	Precision	Recall	F-Measure
C1	98.12 %	97.17 %	97.64 %
C2	83.27 %	99.16 %	90.52 %
C3	99.78 %	84.28 %	91.38 %

The numbers are based on an exhaustive pairwise comparison without clustering.

for this purpose use the CORA dataset again, but this time we use three classifiers with different properties. These are the same three classifiers as used in Reference [12].

Table 15 gives an overview of the classifier quality, achieved by an exhaustive comparison without clustering. The first classifier, *C1*, has both a high precision and a high recall value. The other two classifiers have either only high recall (*C2*) or high precision (*C3*). Note that all three classifiers still lead to good results of the pairwise comparison. The goal of the additional experiments is to evaluate whether the clustering algorithms are resistant to small deviations of the classifier quality. We use again the three pair-selection algorithms from the previous experiments for the pairwise comparison, as well as f-measure, precision, recall, and GMD as measures.

Table 16 shows the results for the different clustering algorithms. For the exhaustive comparison, we can see that the Transitive Closure, Markov clustering, and Merge-Center clustering have a much lower f-measure value for classifier *C2* compared to *C1*, which shows that they are very sensitive regarding the precision value of the classifier. The lower recall value of *C3* does not have such a strong impact on these clustering algorithms. In fact, the Transitive Closure and Merge-Center clustering have significantly higher f-measure value for *C3* compared to *C1*. However, MCC, GCluster, and VOTE/BOEM have a significantly lower f-measure value for *C3*, but for *C2* they show only slight differences. EMCC, Star Clustering and especially GECG show only small divergences for both *C2* and *C3* compared to *C1*.

For Blocking, the results are similar to the exhaustive comparison as pair-selection algorithm. The Transitive Closure and Merge-Center Clustering achieve worse results with *C2* compared with *C1*, but benefit from the higher precision of *C3*. Vice versa, MCC and VOTE/BOEM gain worse results with *C3*, but benefit from the higher recall values of *C2*. The other clustering algorithms have smaller differences for the three classifiers. Interestingly, GCluster this time has a higher difference between *C1* and *C2*, than between *C1* and *C3*.

With the Sorted Neighborhood method as pair-selection algorithm, we can see fewer differences between the three classifiers than for the exhaustive comparison or blocking. Only for the

Table 16. Experimental Evaluation for the Cora Dataset with Three Different Classifiers and Different Pair-selection Algorithms

Exhaustive Comp.	F-Measure in %			Precision in %			Recall in %			GMD		
Clustering Alg.	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
No Clustering	97.64	90.52	91.38	98.12	83.27	99.78	97.17	99.16	84.28	40	52	41
Gold Standard Clust.	99.76	99.81	98.22	100.00	100.00	100.00	99.51	99.61	96.50	12	8	34
MCC	96.04	93.64	80.96	99.34	92.19	99.89	92.96	95.13	68.06	44	58	116
EMCC	97.95	91.22	91.50	99.00	84.63	99.91	96.93	98.93	84.39	33	50	86
GECG	96.84	90.57	92.95	96.71	83.34	99.91	96.98	99.19	86.89	34	49	79
Transitive Closure	88.01	71.63	97.47	78.90	55.92	98.47	99.51	99.61	96.50	40	52	41
GCluster	94.68	92.60	60.45	99.09	95.74	99.92	90.64	89.65	43.33	60	71	84
Markov Clustering	97.34	87.44	95.17	95.27	77.92	98.58	99.51	99.61	91.99	28	43	44
Merge-Center Clust.	89.26	71.63	97.56	80.92	55.92	98.66	99.51	99.61	96.48	39	52	40
Mod. Star Clustering	93.77	87.07	94.49	89.19	78.39	98.50	98.84	97.92	90.80	45	65	58
VOTE/BOEM	89.83	87.85	78.90	98.70	87.51	99.01	82.43	88.20	65.58	48	55	124
Blocking	F-Measure in %			Precision in %			Recall in %			GMD		
Clustering Alg.	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
No Clustering	94.85	91.14	89.13	98.93	89.39	99.77	91.09	92.97	80.54	36	49	41
Gold Standard Clust.	94.85	99.81	98.22	100.00	100.00	100.00	99.51	99.61	96.50	12	8	34
MCC	89.80	90.32	79.38	99.48	97.18	99.89	81.84	84.37	65.86	60	67	128
EMCC	94.98	93.16	88.49	99.10	93.23	99.91	91.19	93.10	79.42	49	61	101
GECG	95.27	90.56	90.46	99.19	88.20	99.91	91.64	93.04	82.64	45	59	94
Transitive Closure	91.30	72.60	97.47	84.34	57.11	98.47	99.51	99.61	96.50	36	49	41
GCluster	67.86	59.58	63.14	98.93	94.92	99.92	51.64	43.42	46.15	75	88	101
Markov Clustering	98.51	90.76	95.14	97.60	83.42	98.60	99.44	99.52	91.92	28	43	45
Merge-Center Clust.	92.17	72.60	97.64	85.84	57.11	98.83	99.51	99.61	96.47	34	49	42
Mod. Star Clustering	93.47	89.11	93.11	93.19	83.84	98.88	93.75	95.08	87.97	50	65	63
VOTE/BOEM	82.96	86.01	68.50	98.64	97.01	99.53	71.59	77.25	52.22	72	68	127
Sorted Neighborh.	F-Measure in %			Precision in %			Recall in %			GMD		
Clustering Alg.	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
No Clustering	55.13	54.88	50.92	98.75	91.60	99.65	38.24	39.17	34.20	31	47	42
Gold Standard Clust.	99.76	99.81	98.19	100.00	100.00	100.00	99.51	99.61	96.43	12	8	36
MCC	32.77	33.25	30.39	99.38	96.15	99.59	19.62	20.10	17.93	120	126	169
EMCC	47.90	48.08	43.03	99.05	94.23	99.71	31.59	32.28	27.43	72	87	130
GECG	49.65	49.95	43.60	99.24	94.88	99.73	33.11	33.90	27.90	63	76	118
Transitive Closure	92.75	77.96	97.62	86.84	64.04	98.83	99.51	99.61	96.43	31	47	42
GCluster	30.72	30.00	27.92	98.72	96.50	99.82	18.19	17.76	16.23	118	143	144
Markov Clustering	73.81	71.56	66.37	98.08	95.26	99.67	59.17	57.31	49.74	28	38	55
Merge-Center Clust.	95.41	77.96	97.67	91.66	64.04	99.01	99.48	99.61	96.37	30	47	44
Mod. Star Clustering	57.30	55.85	55.15	94.66	85.89	99.05	41.09	41.38	38.21	63	96	93
VOTE/BOEM	42.13	41.16	42.09	99.01	96.16	98.83	26.75	26.19	26.74	109	127	159

Transitive Closure and Merge-Center Clustering are the results of *C2* considerably smaller than for *C1*.

The additional experiments show that especially the Transitive Closure, Merge-Center Clustering, and Markov clustering need a classifier with a high precision value to achieve good results. A lower recall value has for these algorithms not necessarily a negative impact. The opposite is true

for MCC and VOTE/BOEM. A lower precision value has a much smaller impact on the f-measure value than a smaller recall value of the classifier. The same is true for GCluster with an exhaustive comparison, but GCluster shows similar results for all three classifiers with blocking or the Sorted Neighborhood Method. EMCC, GECG, and Star Clustering do not show big differences for the three classifiers, so they are not very sensitive for the quality of the classifier. GCluster showed different results for different pair-selection algorithms, thus a high-quality classifier is necessary for GCluster.

7 RELATED WORK

This section provides an overview about related work from three areas. First, we give an overview about related work regarding duplicate detection and its properties in general. Second, we present two papers that evaluate several clustering algorithms for duplicate detection. Finally, we describe alternative approaches for conflict resolution for the result of a pairwise duplicate classification.

Related work regarding Duplicate Detection. Duplicate detection was first defined by Newcombe et al. [28] and has been researched extensively over the past decades. The challenge is to effectively and efficiently identify clusters of records that represent the same real world entity. Recent surveys [8, 13, 26] explain various methods for improving effectiveness and efficiency.

Benjelloun et al. define the ICAR properties (idempotence, commutativity, associativity, and representativity) for match and merge functions in the duplicate detection process [3]. Idempotence means that a record matches itself, whereas commutativity describes whether the order of the records has an impact on the matching result or not. In our evaluation, we use classifiers that fulfill these two properties, leading to an undirected input graph for the clustering algorithms. Without commutativity, we would have a directed input graph. We do not have to consider associativity and representativity, as these are properties of a merge function.

Clustering for Duplicate Detection. Hassanzadeh et al. present the Stringer Duplicate Detection Framework [19]. They use Stringer to evaluate the performance of clustering algorithms for entity resolution. First, they apply a similarity join to retrieve pairs of similar records with their similarity score. They consider only record pairs with a similarity score above a threshold θ. The result of the similarity join is then processed by the clustering algorithm that create clusters of potential duplicates. They use the 29 *Stringer* datasets described in Section 6.2 for their extensive evaluation of the clustering algorithms. Next to key figures like precision, recall, and f-measure, they also use soft criteria (low, medium, high) to present a classification regarding scalability, ability to find the correct number of clusters, robustness against choice of threshold, and robustness against amount and distribution of errors.

Wang et al. also evaluate several clustering algorithms [36]. They formalize the entity resolution problem as a cohesive-based clustering problem on a weighted graph. They present two algorithms, GCluster and HCluster, that solve this problem. GCluster outperforms HCluster on effectiveness, but HCluster is more efficient and better suited for large datasets, as it avoids scanning data frequently.

In comparison to Reference [19], our evaluation contains further datasets, including real-world and larger ones. In comparison to Reference [36], we evaluate more clustering algorithms and also use additional datasets. Compared to both papers, we present and evaluate new clustering algorithms, and particularly evaluate the effect of different pair selection algorithms on the clustering result.

A similar use-case for clustering is entity matching from different sources. Two proposed algorithms are SplitMerge [27] and CLIP [33]. The main difference regarding clustering for duplicate detection is their assumption that each source is already duplicate-free, i.e., for each real-world

entity there exists only one element per source. The following two paragraphs sketch these algorithms. Both use the number of sources as a criterion for the creation of the clusters. Therefore, they cannot be adapted for duplicate detection within a single source. A comparison of both algorithms, especially regarding distributed execution, can be found in Reference [32].

SplitMerge consists of four phases, namely, preprocessing, initial clustering, clustering decomposition, and cluster merge [27]. In the preprocessing phase, the required property values for the similarity calculation are normalized, and the initial clustering phase creates connected components. After this phase, it is still possible that a cluster contains multiple entities of the same source. In this case, some entities are removed based on their similarity to other entities in the cluster, so that only the entity from one source with the highest similarity to the other entities is kept. Cluster decomposition is used to separate entities that have either different or incompatible semantic types, or their similarity to other cluster members is too low. In the end of this phase, a cluster representative is calculated. The last step is cluster merging, which iteratively merges clusters based on the similarity of the previous calculated cluster representatives. Only clusters with less elements than the number of sources are considered.

CLIP uses a similarity graph as input and classifies links between elements as a strong, normal, or weak link [33]. In the first phase, only strong links are considered to determine complete clusters, which are clusters that contain entities from all sources. In the second phase, normal links are also considered and CLIP iteratively clusters the remaining entities based on link priorities, so that no source-inconsistent or overlapping clusters are generated.

Conflict Resolution by using the Crowd. Another possibility to solve conflicts of a pairwise comparison is a manual inspection, e.g., by using the crowd. There are several crowd-sourcing platforms, such as Amazon Mechanical Turk[10] or CrowdFlower,[11] which can be used to improve duplicate detection results. The premise is that humans can solve complex tasks better than computers. Recent research resulted in a variety of algorithms that use crowdsourcing for duplicate detection [37, 39, 42]. The main problems for using the crowd are, on the one hand, the fact that even humans are not always correct with their classification of duplicates or non-duplicates, and on the other hand, the costs for executing classification tasks. Furthermore, for many duplicate detection tasks crowds cannot be used due to data privacy reasons, e.g., health data.

For the first problem, worker quality can be improved by testing the workers before giving them actual tasks. Additionally, tasks can be given to multiple humans and the final classification is the majority answer. The second issue can be tackled by limiting the manual inspections to difficult classifications. If record pairs have a very high or very low similarity, then it is usually unnecessary to pay for a manual inspection. However, manual inspections are also a means to validate the computational classifications. Another approach presented by Wang et al. uses transitive relations to reduce the number of crowd-sourced pairs [38].

Active learning techniques use a small initial dataset to build a classification model that is iteratively optimized by asking a domain-expert to manually inspect record pairs that are difficult to classify. These record pairs are added to the training set and the classification model is rebuilt [8].

8 CONCLUSION

In this article, we have formalized the problem of clustering duplicate detection results and presented several existing and three new clustering algorithms. Our comprehensive experimental evaluation of the clustering algorithms on various datasets and under various algorithm configurations shows that the commonly used Transitive Closure is inferior to most of the other clustering

[10] https://www.mturk.com/mturk/welcome.
[11] https://www.crowdflower.com.

algorithms, especially for the precision of results. In scenarios with small cluster sizes, the choice of the clustering algorithm has no or little effect on the overall result. In scenarios with larger clusters, our proposed EMCC algorithm is, together with Markov Clustering, one of the two best performing clustering approaches for duplicate detection, although its runtime is longer due to the sub-exponential time complexity. EMCC especially outperforms Markov Clustering regarding the precision of the results and additionally has the advantage that it can also be used in scenarios where edge weights are not available.

REFERENCES

[1] Javed A. Aslam, Ekaterina Pelekhov, and Daniela Rus. 2004. The star clustering algorithm for static and dynamic information organization. *J. Graph Algor. Appl.* 8 (2004), 95–129.

[2] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. 2004. Correlation clustering. *Mach. Learn.* 56, 1-3 (2004), 89–113.

[3] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. 2009. Swoosh: A generic approach to entity resolution. *VLDB J.* 18, 1 (2009), 255–276.

[4] Mikhail Bilenko and Raymond J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ACM SIGKDD International Conference of Knowledge Discovery and Data Mining*. 39–48.

[5] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

[6] Peter Christen. 2005. Probabilistic data generation for deduplication and data linkage. In *Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'05)*. 109–116.

[7] Peter Christen. 2011. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.* 24, 9 (2011), 1537–1555.

[8] Peter Christen. 2012. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer, Berlin.

[9] Peter Christen. 2016. Application of advanced record linkage techniques for complex population reconstruction. *Arxiv Preprint Arxiv:1612.04286* (2016).

[10] Xin Dong, Alon Halevy, and Jayant Madhavan. 2005. Reference reconciliation in complex information spaces. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'05)*. 85–96.

[11] Uwe Draisbach and Felix Naumann. 2010. DuDe: The duplicate detection toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB'10)*.

[12] Uwe Draisbach, Felix Naumann, Sascha Szott, and Oliver Wonneberg. 2012. Adaptive windows for duplicate detection. In *Proceedings of the International Conference on Data Engineering (ICDE'12)*. 1073–1083.

[13] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.* 19, 1 (2007), 1–16.

[14] Micha Elsner and Warren Schudy. 2009. Bounding and comparing methods for correlation clustering beyond ILP. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing (ILP'09)*. 19–27.

[15] Jeffrey Fisher and Qing Wang. 2015. Unsupervised measuring of entity resolution consistency. In *Proceedings of the IEEE International Conference on Data Mining Workshop (ICDMW'15)*. 218–221.

[16] Michael R. Garey and David S. Johnson. 197. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York.

[17] Andrey Goder and Vladimir Filkov. 2008. Consensus clustering algorithms: Comparison and refinement. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX'08)*. 109–117.

[18] David Hand and Peter Christen. 2018. A note on using the f-measure for evaluating record linkage algorithms. *Stat. Comput.* 28, 3 (2018), 539–547.

[19] Oktie Hassanzadeh, Fei Chiang, Renée J. Miller, and Hyun Chul Lee. 2009. Framework for evaluating clustering algorithms in duplicate detection. *Proc. Very Large Data Base* 2, 1 (2009), 1282–1293.

[20] Oktie Hassanzadeh and Renée J. Miller. 2009. Creating probabilistic databases from duplicated data. *VLDB J.* 18, 5 (2009), 1141–1166.

[21] Taher H. Haveliwala, Aristides Gionis, and Piotr Indyk. 2000. Scalable techniques for clustering the web. In *Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB'00)*. 129–134.

[22] Melanie Herschel, Felix Naumann, Sascha Szott, and Maik Taubert. 2012. Scalable iterative graph duplicate detection. *IEEE Trans. Knowl. Data Eng.* 24, 11 (2012), 2094–2108.

[23] Roger A. Horn and Charles R. Johnson. 2012. *Matrix Analysis* (2nd ed.). Cambridge University Press, New York.

[24] David Menestrina, Steven Whang, and Hector Garcia-Molina. 2010. Evaluating entity resolution results. *Proc. Very Large Data Base* 3, 1 (2010), 208–219.

[25] Alvaro Monge and Charles Elkan. 1997. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'97)*.
[26] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection (Synthesis Lectures on Data Management)*. Morgan and Claypool Publishers.
[27] Markus Nentwig, Anika Groß, and Erhard Rahm. 2016. Holistic entity clustering for linked data. In *Proceedings of the IEEE International Conference on Data Mining Workshop (ICDM'16)*. 194–201.
[28] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. 1959. Automatic linkage of vital records. *Science* 130, 3381 (1959), 954–959.
[29] Banda Ramadan, Peter Christen, Huizhi Liang, Ross W. Gayler, and David Hawking. 2015. Dynamic sorted neighborhood indexing for real-time entity resolution. *J. Data Info. Qual.* 6, 4 (2015), 15:1–15:29.
[30] Alice Reid, Ros Davies, and Eilidh Garrett. 2002. Nineteenth-century Scottish demography from linked censuses and civil registers. *History Comput.* 14, 1–2 (2002), 61–86.
[31] J. M. Robson. 1986. Algorithms for maximum independent sets. *J. Algor.* 7, 3 (1986), 425–440.
[32] Alieh Saeedi, Markus Nentwig, Eric Peukert, and Erhard Rahm. 2018. Scalable matching and clustering of entities with FAMER. *Complex Syst. Info. Model. Quart.* 16 (2018), 61–83.
[33] Alieh Saeedi, Eric Peukert, and Erhard Rahm. 2018. Using link features for entity clustering in knowledge graphs. In *Proceedings of the European Semantic Web Conference (ESWC'18)*. 576–592.
[34] Robert Endre Tarjan and Anthony E. Trojanowski. 1977. Finding a maximum independent set. *SIAM J. Comput.* 6, 3 (1977), 537–546.
[35] Stijn van Dongen. 2000. *Graph Clustering by Flow Simulation*. Ph.D. Dissertation. University of Utrecht.
[36] Hongzhi Wang, Jianzhong Li, and Hong Gao. 2015. Efficient entity resolution based on subgraph cohesion. *Knowl. Info. Syst.* 46, 2 (2015), 285–314.
[37] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing entity resolution. *Proc. Very Large Data Base* 5, 11 (2012), 1483–1494.
[38] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2013. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'13)*. 229–240.
[39] Sibo Wang, Xiaokui Xiao, and Chun-Hee Lee. 2015. Crowd-based deduplication: An adaptive approach. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'15)*. 1263–1277.
[40] Henry S. Warren Jr.1975. A modification of warshall's algorithm for the transitive closure of binary relations. *Commun. ACM* 18, 4 (1975), 218–220.
[41] Stephen Warshall. 1962. A theorem on boolean matrices. *J. ACM* 9, 1 (1962), 11–12.
[42] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. 2013. Question selection for crowd entity resolution. *Proc. Very Large Data Base* 6, 6 (2013), 349–360.

Received July 2018; revised April 2019; accepted July 2019