University of Toronto, Department of Computer Science, Technical Report No.
CSRG-620

An Evaluation of Clustering Algorithms for Duplicate Detection

Bilal Hussain · Oktie Hassanzadeh · Fei Chiang · Hyun Chul Lee · Renée J. Miller

October 9, 2013

Preamble This project was an offshoot of a PVLDB 2009 paper by Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee and Renée J. Miller [40]. Our idea was to delve deeper into the accuracy of clustering algorithms that have been used for duplicate detection. To do that, we had Bilal Hussain experiment with the algoirthms studied in our original paper (and one or two others) and evaluate them using a larger set of accuracy measures, including those presented by Menestrina et al. [57] in work that followed our initial study [40]. This technical report documents some of the results. The paper itself is a rather jumbled (mostly copied) version of the original PVLDB 2009 paper and we recommend that readers refer to that paper [40], rather than trying to follow the rather awkward, unproofed description in this project report. The experimental section in this technical report is new and produced with the source code from: `https://github.com/hussaibi/libclustER`**. Readers should be warned that we have not been able to replicate the results, so they should take these findings as very preliminary and unverified. This project report can be referenced using the citation: [45].**

Abstract: Duplicate detection, also referred to as *entity resolution* or *record linkage*, is a major challenge in integration and cleaning of large databases. In this paper, we focus on a class of duplicate detection algorithms that rely on clustering a similarity graph. Each node in the similarity graph represents a record and the weight of an edge connecting two nodes reflects the amount of similarity between the corresponding records. The similarity graph can be efficiently constructed using state-of-the-art similarity join techniques. For duplicate detection, a clustering algorithm over the similarity graph is used to produce sets of records that are likely to represent the same entity. In this paper, we present a framework for evaluating the effectiveness of clustering algorithms for duplicate detection. We present the results of our extensive evaluation of a wide range of clustering algorithms. Our evaluation is based on a range of existing and novel quality measures, and reveals a diverse set of behaviors in terms of both accuracy and scalability. Our results show the superiority of a number of clustering algorithms that have not been used in the past for duplicate detection.

B. Hussain · R.J. Miller
Department of Computer Science, University of Toronto
E-mail: hussaibi@cs.toronto.edu, miller@cs.toronto.edu

O. Hassanzadeh
IBM T.J. Watson Research Center
E-mail: hassanzadeh@us.ibm.com

F. Chiang
Department of Computing and Software, McMaster University
E-mail: fchiang@mcmaster.ca

H. C. Lee
Content Understanding & Personalization, LinkedIn
E-mail: culee@linkedin.com

1 Introduction

The presence of duplicate records is a major concern for data quality in large databases. *Duplication detection*, also known as *entity resolution* or *record linkage*, is used to identify records that potentially refer to the same entity. An example of a duplicate detection task would be linking publications to author names based on bibliographic records. Multiple formats, character accents, and typos can making cataloging publications by author names difficult. As such, multiple occurrences of author names referring to the same author would need to be linked, based on some notion of similarity, to properly attribute publications to their respective authors. There are studies and surveys comparing the similarity measures used within these techniques

[19, 39, 42, 28]. However, to the best of our knowledge there are no comprehensive empirical studies that evaluate the quality of the grouping or clustering employed by these techniques. This is the case, despite the large number and variety of clustering techniques that have been proposed for duplicate detection within the information retrieval, data management, theory, and machine learning communities. These clustering algorithms are quite diverse in terms of their properties and their scalability. This diversity provides motivation for a study comparing the effectiveness of the different clustering approaches for the duplicate detection task.

In this paper, we present an evaluation of clustering algorithms for duplicate detection in the context of the Stringer system[1] [41] that provides an empirical framework for understanding what barriers remain towards the goal of truly scalable and general purpose duplication detection algorithms. As shown in Figure 1, Stringer takes as input a set of records and a threshold value. It then finds all pairs of records where their similarity score is above a given threshold value, using a *self-similarity join*. Regarding our earlier example of cataloging publications by authors, the similarity-join would be produced when all author names are compared to one another and a similarity measure is calculated over each pair. The result can be regarded as a weighted graph of records, allowing the application of graph clustering algorithms to group highly similar records. We limit the scope of this paper by considering only clustering algorithms that are referred to as *unconstrained* [71]. These clustering algorithms do not assume as input, the number of entities or other domain-specific parameters. This characteristic conforms to the problem of duplicate detection where the number of entities is not known in advance. In this paper, we use Stringer to understand which clustering algorithms can be used in concert with scalable similarity join algorithms to produce duplicate detection algorithms that are robust with respect to the threshold used for the similarity join, and various data characteristics including the amount and distribution of duplicates.

Past work has focused on duplicate detection assuming complete record comparison information, on using techniques to support declarative semantics, and improving the runtime performance of the duplicate detection task [10]. In contrast, we focus on evaluating performance accuracy (correctness of duplicates detected) when using incomplete record comparison information, on scalable similarity join techniques.

1.1 Contributions

Our contributions include the following.

[1] http://dblab.cs.toronto.edu/project/stringer/

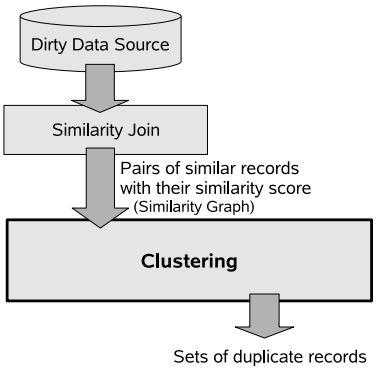


Fig. 1 Duplicate detection framework

- We present a novel classification of graph clustering algorithms and show how they can be used, in conjunction with scalable similarity join algorithms, for duplicate detection. We include highly scalable algorithms together with some more mathematically sophisticated and newer clustering algorithms that have generated a lot of buzz in the data management community, but have not been evaluated for duplicate detection. We also include algorithms from information retrieval as well as graph-theoretic algorithms. The majority of the algorithms presented in this paper were not previously used for duplicate detection. Additionally, we abstract multiple clustering algorithms into a broader subfamily of graph clustering algorithms, which we refer to as Cut-Based Clustering Algorithms.
- We present a comprehensive evaluation studying the behavior of graph clustering algorithms for the task of duplicate detection. Our evaluation is based on a diverse collection of quality measures and on various datasets with many different characteristics. Specifically, we broaden the quality measures to include the K-measure and the Variation of Information.
- We propose and evaluate variants of graph clustering algorithms to improve performance and to analyze inherent nondeterministic behavior in their implementations. These variants are based on a concept derived from social network analysis called *centrality measure*. We propose using centrality measures to approximate existing graph clustering algorithms. To our knowledge, this is the first such evaluation incorporating centrality measures for the purpose of duplicate detection.
- We present an extensive number of performance metrics in our evaluation, and provide an intuition of how they should be interpreted. This allows us to perform an extensive performance and behavior comparison across the different graph clustering algorithms. Specifically, we look at quality evaluation measures for resolving entities and clustering (in general) which lie outside the traditional scope of information retrieval.

– We present the results of our comparison study using a suite of clustering algorithms for duplicate detection over string data. Our results highlight the relationship between the data characteristics, the similarity threshold, and performance accuracy.

This paper is organized as follows. In the next section, we present a brief discussion of related work on duplicate detection and clustering. In Section 3, we describe the clustering algorithms used in our evaluation. In Section 4, we discuss the methodology of our extensive experiments and the characteristics of our datasets. Section 5 presents the set of accuracy measures used in our experiments. Finally, in Section 6 we discuss our experimental results, and conclude in Section 7.

2 Related work

In the context of duplicate detection, several highly scalable similarity join algorithms have been proposed in the past [4, 9, 18, 54, 61]. Several techniques have been proposed to perform effective clustering for duplicate detection. Among these include collective duplicate detection which apply co-occurrence information between two sets of entities (of different types), and probabilistic inference over general networks of different entities, also known as probabilistic relational learning [11, 12, 27]. In our paper, our goal is to evaluate clustering algorithms that use the simplest form of input, that is, the output of a similarity join that includes a set of thresholded similarity scores between pairs of entities. The pairs of entities represent comparison information between entities. We referred to the thresholded similarity join in section 1 as incomplete record comparison information. Hence, in considering the integration of bibliographic databases, our techniques can match tables on publication titles, person names, or any set of attributes about the publications, but will not take advantage information external to the similarity join, such as a social network relationship between the authors. Even when additional information is available, it may not be shared or may be represented differently. We believe that this study, with its strict focus on general purpose techniques, will provide results that can be used to inform empirical studies of more specialized models. The transitivity of the similarity join is then used for detecting duplicated. However, the transitivity can be approximated by using clustering techniques.

A wealth of clustering algorithms have been proposed, including several books [47, 53, 73], surveys [24, 31, 38, 48, 72], and theses [3, 50, 62, 65, 69]. As noted above, our goal is to use clustering algorithms that do not require *a priori* parameter settings (that adjust for cluster properties, data characteristics, or the number of clusters). If such parameters are strictly required, then we include such clustering algorithms in our study if their parameter value(s) remain stable across runtime execution. For example, we consider three algorithms, Markov Clustering (MCL) [65], Affinity-Propagation Clustering (AP) [36], and Cut Clustering (CUT) [32], which all require some parameter input. MCL contains a parameter that influences cluster size, however, the optimal value remains rather stable across applications [71]. . AP contains parameters that influence runtime and exemplar preferences, that can be tuned locally with aggregate functions or globally via a single value. In our experiments, we evaluate the effects of exemplar preference on AP. Finally, CUT contains a single parameter, which as our experiments show, directly influences the quality of the clustering.

In the classification of clustering algorithms, shown in Figure 2, most of the algorithms we consider fall in the partitional class. That is, we are not interested in algorithms that are supervised or only produce hierarchical clusters. We only consider those algorithms that are *unconstrained* [71]. Clustering algorithms like the X-means algorithm [58], which is an extension of K-means, as well as some extensions of popular Spectral clustering algorithms [59, 68] do not require the number of clusters. These algorithms automatically discover the optimal number of clusters usually through some heuristics. We would not consider such algorithms as *unconstrained* per se since the number of clusters is the intrinsic part of the original algorithms that these are built upon and the optimal number of clusters is discovered during post-processing as opposed to during clustering. Moreover, the main application of these algorithms is in pattern recognition and computer vision, with different characteristics that make them unsuitable for our framework. Particularly, the size of the clusters is usually large in these applications whereas for duplicate detection in real world data, there are many small (and even singleton) clusters in the data. This makes some other (unconstrained) clustering algorithms for finding subgraphs [30, 37] inapplicable.

Although some of the algorithms considered in this paper have been evaluated previously on synthetic, randomly generated graphs [15, 65], in document clustering [71], and in computational biology [16], our work is the first to perform an extensive comparative study across all these techniques based on several robust quality measures.

3 Algorithms

We now shift focus to specifying in detail the clustering algorithms considered for grouping duplicates in the Stringer system. In the classification of clustering algorithms in Figure 2 most of our algorithms fall in the partitional class. That is, we are not interested in algorithms that are supervised or only produce hierarchical clusters. All these algorithms

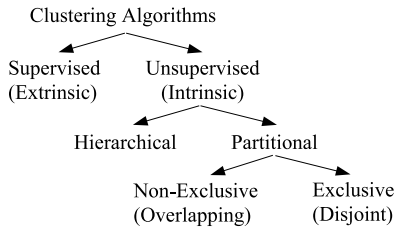


Fig. 2 A classification of clustering algorithms [47].

share the same goal of creating clusters that maximize the intra-cluster weights, and minimize the inter-cluster edge weights. Satisfying this objective is known to be computationally intractable. Thus many approximate solutions either based on heuristics or theoretical justifications have been proposed.

In this section, we describe the clustering algorithms included in our framework. Unconstrained clustering algorithms aim to create clusters containing similar records $\mathcal{C} = \{c_1, \ldots, c_k\}$ where the value of k is unknown [71]. The clustering may be exclusive (disjoint), meaning the base relation is partitioned and there is no record belonging to more than one cluster. Alternatively, non-exclusive clustering permit records to belong to more than one cluster, although it is desirable for this overlap to be small. The Star, Ricochet (OCR, CR), and Articulation Point clustering algorithms may produce overlapping clusters. For the majority of algorithms covered in this section, the input base relation is represented as a graph, $G = (V, E, w)$, instead of a similarity join. In this representation verticies (V) represent records, edges (E) represent a comparison of records, and the similarity between compared records is represented as a weight on the respective edge ($w : E \mapsto \mathbb{R}$). The graph only considers edges with weight above a threshold (θ), meaning the algorithms are given incomplete record comparison information. For the Affinity Propagation and Markov Clustering algorithms, the input relation is represented as an adjacency matrix. These are the only alternative representations used in this paper.

3.1 Single-pass Algorithms

In single-pass algorithms, intermediate results are not materialized into memory for further processing. All the algorithms can be efficiently implemented by a single scan of the list of similar pairs returned by the similarity join (although some require the list to be sorted by similarityscore). Note that the construction of the similarity join is not considered a part of the single-pass algorithms. Figure 3 illustrates the result of applying these algorithms to a sample similarity graph [41].

3.1.1 Partitioning (Transitive Closure)

Partitioning based algorithms have been applied in early duplicate detection work [29, 44]. The algorithm performs clustering by first assigning each node to its own cluster. The output of the similarity join showing the list of similar entity pairs is scanned once. If two entities are similar, but their nodes are not in the same cluster, then their clusters are merged [44, 26]. Figure 3(a) shows a sample clustering, whereby many records that are not similar, may be grouped into the same cluster.

3.1.2 CENTER

The Center clustering algorithm (CENTER) [43] was originally proposed for retrieval of web documents. CENTER performs clustering by partitioning the similarity graph into clusters that have a *center*, and all records in each cluster are similar to their respective center. The algorithm requires the list of similar pairs to be sorted by decreasing order of similarity scores. The algorithm then performs clustering by a single scan of the sorted list. As the sorted list is traversed, one node is arbitrarily labeled as a center and the other non-center. These labels remain fixed for all subsequent node considerations. This process is similar to using disjoint-sets to compute a transitive-closure. Figure 3(b) illustrates how this algorithm clusters a sample graph of records. In this figure, node u_1 is in the first pair in the sorted list of similar records and node u_2 appears in a pair right after all the nodes similar to u_1 are visited, and node u_3 appears after all the nodes similar to u_2 are scanned. As the figure shows, this algorithm could result in more clusters than Partitioning since it assigns to a cluster only those records that are similar to the center of the cluster.

3.1.3 Merge Center

The Merge-Center algorithm (MC) is a simple extension of the CENTER algorithm [41]. MC performs similar to CENTER, but merges two clusters c_i and c_j whenever a record similar to the *center* node of c_j is in the cluster c_i, i.e., a record that is similar to the center of cluster c_j is similar to the center of c_i. This is done by scanning the list of the similar records once, but keeping track of the records that are already in a cluster. Again, the first time a node u appears in a pair, it is assigned as the center of the cluster. All the subsequent nodes v that appear in a pair (u, v), and are not assigned to any cluster, are assigned to the cluster of u. Whenever a pair (u, v') is encountered such that v' is already in another cluster, the cluster of u is merged with the cluster of v'. Figure 3(c) shows the resulting clusters, assuming that the nodes u_1, u_2 and u_3 are the first three nodes in the sorted list of similar records that are assigned as the center

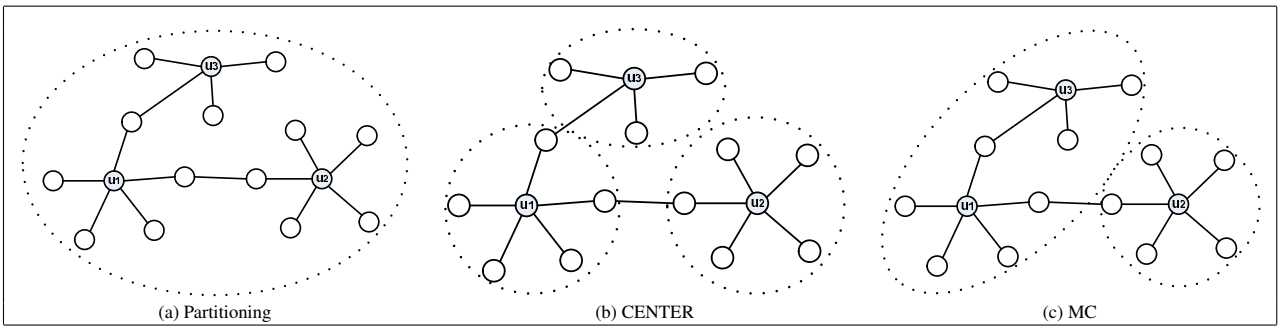


Fig. 3 Illustration of single-pass clustering algorithms

Algorithm 1 CENTER algorithm

Require: An undirected weighted graph, $G = (V, E, w)$
$S = sort\,(E, w\,(E))$
$center = \emptyset$ {is a set}
$noncenter = \emptyset$ {is a set}
$cluster = \emptyset$ {a map from vertices to vertex sets}
for (u, v) in S **do**
  if $u \neq v$ **then**
    if $u \notin center$ and $u \notin noncenter$ and $v \notin center$ and $v \notin noncenter$ **then**
      $center.add(u)$ {Status assigned arbitrarily. u did not have to become a center.}
      $noncenter.add(v)$
      $cluster\,[u]\,.add(\{u, v\})$
    else if ($u \in center$ and $v \in center$) or ($u \in noncenter$ and $v \in noncenter$) **then**
      ignore
    else if $v \in center$ and $u \notin noncenter$ **then**
      $noncenter.add(u)$ {So u was unlabeled}
      $cluster\,[v]\,.add(\{u\})$
    else if $u \in center$ and $v \notin noncenter$ **then**
      $noncenter.add(v)$ {So v was unlabeled}
      $cluster\,[u]\,.add(\{v\})$
    else
      ignore
    end if
  end if
end for
return $cluster\,[center] \cup \{\{v\} \mid v \in V, v \notin center \wedge v \notin noncenter\}$ {collect clusters and singletons}

Algorithm 2 MC algorithm

Require: An undirected weighted graph, $G = (V, E, w)$
$S = sort\,(E, w\,(E))$
$center = \emptyset$ {is a set}
$noncenter = \emptyset$ {is a set}
$cluster = \emptyset$ {a disjoint-set data structure for building clusters}
for (u, v) in S **do**
  if $u \neq v$ **then**
    if $u \notin center$ and $u \notin noncenter$ and $v \notin center$ and $v \notin noncenter$ **then**
      $center.add(u)$ {Labels given arbitrarily.}
      $noncenter.add(v)$
      $cluster.MakeSet(u)$
      $cluster.MakeSet(v)$
      $cluster.Union(u, v)$ {assume first vertex, u, becomes representative of union}
    else if $u \in center$ and $v \in center$ **then**
      $cluster.Union(u, v)$
    else if ($u \in noncenter$ and $v \in noncenter$) **then**
      ignore
    else if $v \in noncenter$ and $u \in center$ **then**
      $cluster.Union(u, cluster.Find(v))$ {union of u , and v 's representative (a center)}
    else if $u \in noncenter$ and $v \in center$ **then**
      $cluster.Union(v, cluster.Find(u))$ {union of u , and v 's representative (a center)}
    else if $v \in center$ **then**
      $noncenter.add(u)$ {So first time seeing u}
      $cluster.MakeSet(u)$
      $cluster.Union(v, u)$
    else if $u \in center$ **then**
      $noncenter.add(v)$ {So first time seeing v}
      $cluster.MakeSet(v)$
      $cluster.Union(u, v)$
    else
      ignore
    end if
  end if
end for
return $clusters.AllSets()$

of a cluster. As Figure 3(c) shows, MC creates fewer clusters than the CENTER algorithm, but more than the Partitioning algorithm.

3.2 Cut-based Algorithms

We introduce the notion of Cut-based clustering algorithms which utilize cuts to generate clusters. A cut partitions a similarity graph's vertices into two mutually exclusive sets. The Star and Cut clustering algorithms follow this intuition by using first-order neighborhoods (defined in Section 3.2.3) and minimum cuts (defined in Section 3.2.1), respectively. The sequential variants of the Ricochet family of clustering algorithms traverse all vertices, and apply a Kernighan-Lin (KL) [49] heuristic to partition the graph. This is done by swapping the vertices between partitions to minimize the inter-cluster edge weights. All the algorithms in this section provide a solution to the cut function used in Algorithm 3. Note that cuts can be calculated in a variety of ways, de-

pending on the the optimization and objective functions used to determine the cut. Additionally, the ordering used to traverse the vertices changes the cuts used to form clusters.

Algorithm 3 A General Cut-Based Clustering algorithm

Require: An undirected weighted graph, $G = (V, E, w)$
  $S = asList(V)$ {S is a list of vertices.}
  $clusters = \emptyset$ {a map from vertices to vertex sets}
  $marked = \emptyset$
  $verextRef = \emptyset$ {used to grab at clusters}
  repeat
    $s = pop\,(S)$
    if $s \notin marked$ **then**
      if overlapping clusters are allowed **then**
        $someCut = cut\,(s, G, \dots)$ {cut returns the partition containing s after performing the cut. $\dots$ are place holder for additional arguments that may be used to compute cut}
        if duplicate clusters or subsets of clusters are not allowed **then**
          $G = removeVertex\,(s, G)$
        end if
      else
        $someCut = cut\,(s, G, \dots)$
        $someCut = someCut - marked$
        $S = S - someCut$
      end if
      $marked = marked \cup someCut$
      $vertexRef = vertRef \cup s$
      $clusters\,[s]\,.add\,(someCut)$
    end if{This is where the sequential Ricochet algorithms would apply the KL-heuristic. Overlapping clusters are transformed into proper partitions with the heuristic.}
  until $(marked = V) \vee (S = \emptyset)$
  return $clusters[vertexRef]$

3.2.1 Cut Clustering

Given a directed graph $G = (V, E)$ with edge capacities $c(u, v) \in \mathbf{Z}^+$, and two vertices s, t, the $s-t$ maximum flow problem is to find a maximum flow path from the source s to the sink t that respects the capacity constraints.[2] Intuitively, if the edges are roads, the max flow problem determines the maximum flow rate of cars between two points. The *max flow-min cut* theorem proven by Ford and Fulkerson [33] states that finding the maximum flow of a network is equivalent to finding the minimum cut that separates s and t. Specifically, this involves finding a non-trivial bipartition of the vertices , where s and t are in different partitions, such that the cut weight (the sum of edge weights crossing between the bipartition) is minimal. There are many applications of this theorem to areas such as network reliability theory, routing, transportation planning, and cluster analysis.

We implemented and evaluated the *Cut Clustering* algorithm based on minimum cuts proposed by Flake, Tarjan, and Tsioutsiouliklis [32]. The goal is to find clusters with small inter-cluster cuts so that the intra-cluster weights are maximized giving strong connections within the clusters. The algorithm is based on inserting an artificial sink t into G and finding the minimum cut between each vertex $u \in U$ (the source) and t. Removing the edges in the minimum cut yields two sets of clusters. Vertices participating in a cluster are not considered as a source in subsequent evaluations. Multiple iterations of finding minimum cuts yields a minimum cut tree, and after removing the sink t, the resulting connected components are the clusters of G. The steps for Cut Clustering are demonstrated by Algorithm 4. The Cut Clustering algorithm contains a parameter α that defines the weight for edges connected to the sink t. We select a suitable α value for our experiments as described further in Section 4.3. Algorithm 4 defines the cut function for Cut clustering, showing its relationship with Algorithm 3.

Algorithm 4 cut function for Cut Algorithm

Require: An undirected weighted graph, $G = (V, E, w)$
Require: $source \in V$
Require: Default edge weight α
  $E_1 = \{(u, sink) \mid \forall u \in V\}$ {where $sink$ is a temporary vertex}
  $w\,(E_1) = \alpha$
  $E_1 = E \cup E_1$
  $V_1 = V \cup \{sink\}$
  $G_1 = (V_1, E_1)$
  $(S, T) = minimumCut\,(G_1, source, sink)$
  {(S, T) is a partition of vertices, where $source \in S$, $sink \in T$ and $S \cap T = \emptyset$}
  return S

3.2.2 Articulation Point Clustering

An articulation point is a vertex whose removal (together with its incident edges) makes the graph disconnected. A graph is biconnected if it contains no articulation points. A biconnected component of a graph is a maximal biconnected graph. Finding biconnected components of a graph is a well-studied problem that can be performed in linear time [20]. The 'removal'[3] of all articulation points separates the graph into biconnected components. This algorithm is based on a scalable technique for finding articulation points and biconnected components in a graph. Given a graph G, the algorithm identifies all articulation points in G and returns all vertices in each biconnected component as a cluster. Note that overlapping clusters are produced. Figure 4 shows an example. A depth-first search traversal is used to find all articulation points and biconnected components in G. We refer the reader to Bansal et al. for details of a scalable and memory efficient implementation of the algorithm and its

[2] Undirected graphs are modeled with bi-directional edges.

[3] Descriptive terminology. The articulation points are not actually removed. These vertices serve as links between the biconnected components and participate in each incident biconnected component vertex set.

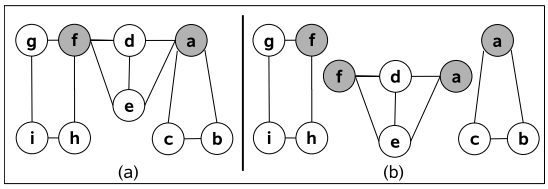


Fig. 4 (a) Articulation points are shaded, (b) Biconnected components.

pseudo-code [7]. We provide the pseudo-code for the cut function associating Articulation Point Clustering with Cut-based Clustering in Algorithm 5. Algorithm 5 defines the cut function solution for articulation point clustering, showing how it fits with the cut based clustering given by Algorithm 3.

Algorithm 5 *cut* function for Articulation Point Algorithm

Require: An undirected weighted graph, $G = (V, E, w)$
Require: $source \in V$
$dfstree = search_{depthfirst}(G, source)$
Identify all articulation points in $dfstree$
remove descendants of all articulation points from $dfstree$
$S = vertices(dfstree)$ {The implied cut is $(S, V - S)$ }
return S

3.2.3 Star Clustering Algorithm

This algorithm is motivated by the fact that high-quality clusters can be obtained from a weighted similarity graph by: (1) removing edges with weight less than a threshold θ, and (2) finding a *minimum clique cover* with maximal cliques on the resulting graph. This approach ensures that all the nodes in one cluster have the desired degree of similarity. Furthermore, minimal clique covers with maximal cliques allow vertices to belong to several clusters, which is a desirable feature in many applications (including ours). Unfortunately this approach is computationally intractable. It is shown that the clique cover problem is NP-complete and does not even admit polynomial-time approximation algorithms [64]. The Star clustering algorithm [5] is proposed as a way to cover the graph by *dense star-shaped subgraphs* instead. Aslam et al. [5] prove several interesting accuracy and efficiency properties, and evaluate the algorithm for document clustering in information retrieval. The cut function solution associating the Star clustering algorithm and the cut-based algorithm is provided in Algorithm 6. The first-order neighborhood of a vertex in a graph, $N_V(G, v)$, can be defined as the set of vertices incident to the queried vertex, v. The degree of a vertex referrers to the number of edges or vertices incident to the queried vertex, $|N_V(G, v)|$.

Algorithm 6 *cut* function for Star algorithm

Require: An undirected weighted graph, $G = (V, E, w)$.
Require: A vertex $v \in V$
$S = N_V(G, v) \cup \{v\}$ {The implied cut is $(S, V - S)$ }
return S
Ensure: The returned value is a set of vertices.

3.2.4 The Ricochet Family of Algorithms

Wijaya and Bressan propose a family of unconstrained algorithms called 'Ricochet' due to their strategy resembling the rippling of stones thrown in a pond [71]. These algorithms perform clustering by alternating between two phases. In the first phase, the seeds of the clusters are specified, which is similar to selecting star centers in the Star algorithm. In the second phase, vertices are assigned to clusters associated with seeds. This phase is similar to the re-assignment phase in the K-means algorithm. Wijaya and Bressan propose four versions of the algorithm. In two of the algorithms, seeds are chosen sequentially one by one, while in the two other algorithms seeds are chosen concurrently. The sequential algorithms produce disjoint clusters, whereas concurrent algorithms may produce overlapping clusters. In all four algorithms, a weight is associated with each vertex . We briefly describe the four algorithms below and refer the reader to Wijaya and Bressan for the complete algorithm pseudo-code. The re-assignment phase, where vertices are swapped between partitions, is equivalent to using the Kernighan-Lin (KL) [49] heuristic. Note that only The sequential Ricochet algorithms lend themselves to the Cut-based clustering algorithm.

Sequential Rippling (SR) performs clustering by first sorting the nodes in descending order of their weight (average weight of their adjacent edges). New seeds are chosen one by one from this sorted list. When a new seed is added, vertices are re-assigned to a new cluster if they are closer to the new seed than they were to their original seed. If there are no re-assignments, then no new cluster is created. If a cluster is reduced to a singleton, it is reassigned to its nearest cluster. The algorithm stops when all nodes are considered.

Clusters are formed by cutting the first-order neighborhood of the seed (same as in Algorithm 6). These clusters overlap before re-assignments are applied. The re-assignments phase swaps records, effectively removing overlaps. The re-assignments phase is an application of the KL-heuristic.

Balanced Sequential Rippling (BSR) is similar to the sequential rippling when selecting the first seed, and has a similar second phase. However its first phase differs whereby it chooses the next seed to maximize the ratio of its weight to the sum of its similarity to the seeds of existing clusters. This strategy is employed to select a node with

a high weight that is far enough from the other seeds. BSR conforms to Cut-based clustering.

Concurrent Rippling (CR) initially marks every vertex as a seed. In each iteration, the algorithm picks for each seed the edge with highest weight. If the edge connects the seed to a vertex that is not a centroid, the vertex is assigned to the same cluster as the seed, and the seed is assigned centroid status. If the vertex is a centroid, it is assigned to the cluster of the other seed only if its weight is smaller than the weight of the seed. The heavier of the two wins centroid status. This iteration (propagation of ripple) is performed equally across all seeds. This requires sorting the edges in descending order of their weights, finding the minimum value of the weight of the edges picked in each iteration of the algorithm, and processing all the edges that have a weight above the minimum weight value. CR *does not* conform to Cut-based clustering.

Ordered Concurrent Rippling (OCR) performs clustering similar to concurrent rippling but removes the requirement that the rippling propagates at equal speeds (i.e. processing all the edges that have a weight above the minimum weight value). Therefore this algorithm is relatively more efficient and also could possibly create higher quality clusters by favoring heavy seeds. OCR *does not* conform to Cut-based clustering.

3.3 Correlation Clustering

Suppose we have a graph G on n nodes, where each edge (u, v) is labeled either $+$ or $-$ depending on whether u and v have been deemed to be similar or different. Correlation clustering, originally proposed by Bansal et al. [6], refers to the problem of producing a partition (a clustering) of G that agrees as much as possible with the edge labels. More precisely, correlation clustering solves a maximization problem where the goal is to find a partition that maximizes the number of $+$ edges within clusters and the number of $-$ edges between clusters. Similarly, correlation clustering can also be formulated as a minimization problem where the goal is to minimize the number of $-$ edges inside clusters and the number of $+$ edges between clusters.

Correlation clustering is a *NP-hard* problem [6]. Thus, several attempts have been made to approximate both the maximization and minimization formulations [6, 17, 25, 63]. Most of them are different ways of approximating its linear programming formulation. For the maximation formulation, Bansal et al. give a polynomial time approximation scheme. For the minimization formulation, Bansal et al. give a constant factor approximation. They also present a result which states that any constant factor approximation for the minimization problem in $\{+, -\}$-graphs can be extended as a constant factor approximation in general weighted graphs. For the original purpose of our application, we had implemented and evaluated the algorithm Cautius [6]. Using a notion of "δ-goodness", the algorithm Cautius expands a cluster associated with an arbitrary node by adding its neighbors that are δ-good into the cluster while removing its neighbors that are δ-bad from the given cluster.

In later work, Ailon et al. proposed a better approximation scheme for the minimization formulation of correlation clustering [2]. The proposed algorithm *CC-Pivot* is equivalent to the partition variant of *STAR* algorithm, where the algorithms is applied only to $+$ edges. Unlike *STAR*, *CC-Pivot* uses a randomized vertex ordering, as opposed to an ordering based on vertex degree. Note that *CC-Pivot* conforms to Cut-based clustering (see Algorithm 3). Therefore, we study the *CC-Pivot* algorithm in our paper.

3.4 Probabilistic Clustering

The probability-based algorithms are clustering algorithms we categorize as either using a graphical-model for learning (such as factor-graphs) or by treating the graph as the setup for simulating stochastic processes.

3.4.1 Markov Clustering (MCL)

The Markov Cluster Algorithm (MCL), proposed by Stijn van Dongen [65], is an algorithm based on simulation of (stochastic) flow in graphs. MCL clusters the graph by performing random walks on a graph using a combination of simple algebraic operations on its associated stochastic matrix. Similar to other algorithms considered in our paper, it does not require any priori knowledge about an underlying cluster structure. The algorithm is based on a simple intuition that a region with many edges inside forms a cluster and therefore the amount of flow within a cluster is strong. On the other hand, there exist a few edges between such produced regions (clusters) and therefore the amount of flow between such regions (clusters) is weak. Random walks (or flow) within the whole graph are used to strengthen flow where it is already strong (e.g. inside a cluster), and weaken it where it is weak (e.g. between clusters). By continuing with such random walks an underlying cluster structure will eventually become visible. Therefore, such random walks end when we find regions (clusters) with strong internal flow that are separated by boundaries with hardly any flow.

$$expansion(M) = M \times M \tag{1}$$

$$inflation(M = [m_{i,j}]_{n,n}, r) = \left[\frac{m_{i,j}^r}{\sum_{i=1}^{n} m_{i,j}^r} \right]_{n,n} \tag{2}$$

The flow simulation in the MCL algorithm is as an alternate application of two simple algebraic operations on stochastic matrix associated with the given graph. The first algebraic

operation is called *expansion*, which coincides with normal matrix multiplication of a random walk matrix. Expansion models the spreading out of flow as it becomes more homogeneous. The second algebraic operation is called *inflation*, which is a Hadamard power followed by a diagonal scaling of another random walk matrix. Inflation models the contraction of flow, becoming thicker in regions of higher current and thinner in regions of lower current. The sequential application of expansion and inflation causes flow to spread out within natural clusters and evaporate in between different clusters. By varying the inflation parameter of the algorithm, clusterings on different scales of granularity can be found. Therefore, the number of clusters cannot and need not be specified in advance, allowing the algorithm to be adapted to different contexts. Algorithm 7 provides further details.

Algorithm 7 MCL Algorithm (without any numerical approximations)

Require: An undirected weighted graph, $G = (V, E, w)$
Require: A real value, $r > 1$, controlling the rate of inflation (granularity of resultant clusterings)
$M_1 = graphToAdjacencyMatrix(G)$
$M_1 = inflation\,(M_1^t, 1)^t$ {Normalize rows to treat as stochastic matrix.}
repeat
$M_2 = expansion(M_1)$
$M_1 = inflation(M_2, r)$
until M_2 and M_1 no longer change based on some notion of convergence (i.e. $M_1 - M_2 = 0$)
$output = matrixToGraph(M_1)$
$output = transitiveClosure(output)$
return $output$

3.4.2 Affinity-Propagation Clustering

The Affinity-Propagation (AP) [36] clustering algorithm uses factor-graph inference and *belief propagation* to form clusters. The update rules used in belief propagation allow AP to scale for distributed applications. Additionally, AP uses a different system of operations for probability updates, avoiding the usage of matrix products. Theoretically, AP is easy to scale and parallelize. We consider AP for string duplicate detection because it has been considered previously for sentence clustering, and previously compared and related conceptually [67] to Markov Clustering within a bioinformatics context [16].

AP takes a similarity-matrix as input along with some additional parameters. These additional parameters specify convergence conditions, vertex preferences, and algorithm termination. AP iterates through two consecutive phases (similar to MCL). The first phase updates a vertex's responsibility to another vertex. Responsibility calculates how well a vertex can represent another. The representative vertex is known as an *exemplar*. The second phase updates a vertex's availability to an exemplar. Availability calculates how appropriate it is for a vertex to be represented by another vertex. Convergence is decided based on the combination of availabilities and responsibilities. Non-zero values of the combination indicate exemplars. All non-exemplars are assigned to exemplars based on maximum similarity (in a manner similar to CENTER, but with centroid and non-centroid status pre-assigned).

Algorithm 8 Calculating/Updating Responsibilities

Require: A matrix, $S = [s_{i,j}]_{n,n}$, of similarities
Require: A matrix, $A = [a_{i,j}]_{n,n}$, of availabilities
Require: A row entry, i
Require: A column entry, k
return $s_{i,k} - \max_{z \neq k} (a_{i,z} + s_{i,z}, -\infty)$

Algorithm 9 Calculating/Updating Availabilities

Require: A matrix, $S = [s_{i,j}]_{n,n}$, of similarities
Require: A matrix, $R = [r_{i,j}]_{n,n}$, of responsibilities
Require: A row entry, i
Require: A column entry, k
if $i = k$ **then**
return $\sum_{z \neq k} r(z, k)$
else
return $\min\left(\sum_{z \neq i} r(z, k), 0\right)$
end if

Exemplar Preferences. The vertex preferences in AP act as a prior belief of how likely exemplar status will be set for a vertex. The preferences are expected to be in the same units as the similarities in the input matrix. Preferences are inserted into the similarity-graph as self-looping edges. We consider four variants of preference in this paper. The first is referred to as *uniform preference (uAP)*, where every vertex is given the same constant as a preference value.

The second variant we referred to as *neighborhood-mean preference (μAP)*. Here the preferences are set as the mean weight for a vertex's neighborhood of edges, $N_E(v \in V, G)$. This is the only variant which sets preferences for every vertex. The preferences change according to the θ setting used for Stringer's similarity-join. Completely disconnected vertices are given the maximum possible similarity-measure as a preference.

$$\mu AP(v \in V) = \left(\frac{\sum_{e \in N_E(v,G)} sim(e)}{|N_E(v,G)|} \right) \quad (3)$$

The last variants awe call *sparse-median (mAP)*, and *sparse-minimum (θAP)*. Sparse-minimum sets the preference of ev-

ery vertex to the θ threshold, the minimum edge-weight among all non-zero edge-weights. Sparse-medium sets the preference of every vertex to the median edge-weight among all non-zero edge-weights. For large θ values, both methods should produce similar preferences.

Algorithm 10 Affinity Propagation

Require: A similarity graph, $G = (V, E, w = sim)$
Require: A vector of preferences, $pref$
  $S = graphToMatrix(G)$
  $diag(similarities) = pref$
  $R = A = 0_{|V|,|V|}$
  repeat
    $R = updateResponsibilities(S, A)$
    $A = updateAvailabilities(S, R)$
    {Assume updates calculated for all matrix index pairs.}
    {Matrix index pair traversal and parameter input omitted here.}
    $pseudoMarginals = R + A$
    $exemplars = index(pseudoMarginals > 0)$
  until some notion of convergence, or a maximum allowable number of iterations has been reached
  return proper assignments of non-exemplars to $exemplars$ based on maximum similarity

4 Evaluation

In this section, we present our evaluation methodology. We first describe the data characteristics used in our experiments. We then explain the experimental settings including the similarity measure used for calculating the approximate join.

4.1 Datasets

We generated data using an enhanced version of the UIS database generator which has been effectively used in the past to evaluate duplicate detection algorithms and has been made publicly available [39, 44]. We used the data generator to inject different realistic types and percentages of errors to a clean database of string attributes. The erroneous records made from each clean record were put in a single cluster (which we used as a ground truth) in order to be able to measure the quality of the clustering algorithms. The generator permits the creation of data sets of varying sizes, error types and distributions. Different error types injected by the data generator include common edit errors (insertion, deletion, replacement or swapping of character)[4] token swap errors and domain specific *abbreviation errors*, e.g., replacing `Inc.` with `Incorporated` and vice versa. We used two different clean sources of data: a data set consisting of *company names* that contains $2,139$ records (names of companies) with average record length of 21.03 characters and 2.92 words in each record on average, and a data set consisting of titles from *DBLP* which contains $10,425$ records with an average of 33.55 characters and 4.53 words in each record. Note that the *data sets created by the data generator can be much larger than the original clean sources*. In our experiments, we created data sets of up to $100K$ records.

For the results in this paper, we used 29 different datasets[5] (tables) with different sizes, error types and distributions. Tables 1 and 2 show the description of all these datasets along with the percentage of erroneous records in each dataset (i.e., the average number of the records in each cluster which are erroneous), the percentage of errors within each duplicate (i.e., the number of errors injected in each erroneous record), the percentage of token swap and abbreviation errors as well as the distribution of the errors (column Dist. in Table 2), the size of the datasets (the number of records in each table) and the number of the clusters of duplicates (column Cluster# in Table 2).[6] Five datasets contain only a single type of error (3 levels of edit errors, token swap or abbreviation replacement errors) to measure the effect of each type of error individually. The datasets with uniform distribution have equal cluster sizes on average (e.g., 10 records in each cluster on average for a dataset of $5,000$ records with 500 clusters) whereas the size of the clusters in the Zipfian datasets follow a Zipfian distribution (i.e., most of the clusters have size 1 while a few clusters are very large). The Zipfian distribution is normally used to model the occurrence of rare events (in this case errors). It is also used to model the characteristics of a linguistic or world-wide-web-harvested corpus. We expect the task of duplicate-detection may be used with some sort of harvesting process, and as such use the Zipfian distribution to generate a more realistic dataset. We believe the errors in these datasets are highly representative of common types of errors in databases with string attributes [39].

4.2 Centrality Measures

The importance a vertex has within a graph can be expressed by the notion of centrality. Centrality has traditionally been used in analyzing social networks [60]. The effects of sampling graph edges and edge connectivity errors on different centralities have been studied for the past decade [21, 14, 35].

Centrality measures determine the relative importance of a vertex in a graph structure, and its relationship to neighboring vertices. Centrality measures have been studied in social

[4] These errors are injected based on a study of common types of edit errors found in real dirty databases [44].

[5] The datasets refer to the output of the data generator, not the clean data sources.

[6] All these datasets along with a small sample of them are available at: http://dblab.cs.toronto.edu/project/stringer/clustering/

Table 1 Datasets used in the experiments

Group	Name	Percentage of			
		Erroneous Duplicates	Errors in Duplicates	Token Swap	Abbr. Error
High Error	H1	90	30	20	50
	H2	50	30	20	50
Medium Error	M1	30	30	20	50
	M2	10	30	20	50
	M3	90	10	20	50
	M4	50	10	20	50
Low Error	L1	30	10	20	50
	L2	10	10	20	50
Single Error	AB	50	0	0	50
	TS	50	0	20	0
	EDL	50	10	0	0
	EDM	50	20	0	0
	EDH	50	30	0	0
Zipfian High	ZH1	90	30	20	50
	ZH2	50	30	20	50
Zipfian Medium Error	ZM1	30	30	20	50
	ZM2	10	30	20	50
	ZM3	90	10	20	50
	ZM4	50	10	20	50
Zipfian Low	ZL1	30	10	20	50
	ZL2	10	10	20	50
DBLP High	DH1	90	30	20	0
	DH2	50	30	20	0
DBLP Medium Error	DM1	30	30	20	0
	DM2	10	30	20	0
	DM3	90	10	20	0
	DM4	50	10	20	0
DBLP Low	DL1	30	10	20	0
	DL2	10	10	20	0

Table 2 Size, distribution and source of the datasets

Group	Source	Dist.	Size	Cluster#
High Error, Medium Error, Low Error, Single Error	Company Names	Uniform	5K	500
Zipfian High Zipfian Medium Zipfian Low	Company Names	Zipfian	1.5K	1K
DBLP High DBLP Medium DBLP Low	DBLP Titles	Uniform	5K	500

network analysis to identify authoritative hubs that influence the information flow and structure of a social network [51]. Similarly, the Star clustering algorithm adopted centrality measures to assess relative vertex importance when selecting star centers, a choice that directly influence the quality of the final clustering [70]. In the context of Cut-based clustering algorithms, centrality allows us to order the vertices prior to running Algorithm 3. Specifically, we use centralities to analyze the deterministic behavior of the algorithms. In our paper, we consider centrality measures as defined and used previously for the Star clustering algorithms [70], which are further detailed below.

In our work, we use clustering to maximize intra-cluster similarity (i.e., the average similarity among pairs of vertices within a cluster) and minimize inter-cluster similarity (i.e., the similarity among pairs of edges across different clusters), such that vertices representing similar strings belong to the same cluster. To improve the effectiveness of our clustering algorithms, we apply a set of centrality measures, as described below, that consider the edge weight similarity scores between two vertices.

Markov Steady-State. This centrality measure indicates the probability other vertices will associate with a given vertex. This probability is calculated by assuming random walks occur along the edges of the similarity graph and the similarity indicates the likelihood of walking between vertices. The computation begins by interpreting the similarity-graph as an adjacency matrix and normalizing all rows, producing a stochastic matrix. The stochastic matrix is then multiplied with itself repeatedly until the resulting matrix no longer changes or meets sufficient convergence conditions. The centrality measure is the column sum of the resulting matrix.

Degree. The degree of a vertex is the number of adjacent vertices.

Strength. The strength of a vertex is the sum of all weights (similarities) along its (first-order) neighborhood.

Mean. The mean centrality measure is obtained by dividing the vertex strength by its degree.

We evaluate these centralities using the Star, Center, and Affinity-Propagation Clustering algorithms. However, the semantic requirement of the Affinity Propagation Algorithm for setting exemplar preferences limits the type of usable centrality measures to Mean (see Section 3.4.2).

4.3 Settings

4.3.1 Similarity Function

There are a large number of similarity measures for string data that can be used in the similarity join. Based on past work comparing such measures [42], we use weighted-Jaccard similarity along with q-gram tokens (substrings of length q of the strings) as the measure of choice due to its relatively high efficiency and accuracy compared with other measures. Jaccard similarity is the fraction of tokens in r_1 and r_2 that are present in both.

$$sim_{Jaccard}(r_1, r_2) = \frac{|r_1 \cap r_2|}{|r_1 \cup r_2|} \tag{4}$$

Weighted-Jaccard similarity is the weighted version of Jaccard similarity, i.e.,

$$sim_{WJaccard}(r_1, r_2) = \frac{\sum_{t \in \mathbf{r}_1 \cap \mathbf{r}_2} w(t, R)}{\sum_{t \in \mathbf{r}_1 \cup \mathbf{r}_2} w(t, R)} \tag{5}$$

where $w(t, R)$ is a weight function that reflects the importance of the token t in the relation R. We choose the commonly-used Inverse Document Frequency (IDF) as token weights. IDF is the logarithm of the inverse of the frequency with which a token occurs over the base relation.

$$w(t, R) = -log\left(\frac{n_t}{N}\right) \tag{6}$$

For our evaluations, we used a slight modification based on the RSJ (Robertson/Sparck Jones) weights which has been shown to make the weight values more effective [39]. Here N is the number of tuples in the base relation R and n_t is the number of tuples in R containing the token t.

$$w_{RJS}(t, R) = log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right) \quad (7)$$

The similarity value returned is between 0 (for strings that do not share any q-grams) and 1 (for equal strings).

Note, the similarity predicate can be implemented declaratively and used as a join predicate in a standard RDBMS engine [39], or used with some of the specialized, high performance, state-of-the-art approximate join algorithms [4, 9]. In our experiments, we use q-grams of size 2. We use a q-gram generation technique proposed in previous work [39]. In this generation technique, strings are first padded with whitespace at the beginning and the end, then all whitespace is replaced with $q - 1$ special symbols (e.g., \$). For example, the string "lorem ipsum" is padded with whitespace becoming " lorem ipsum ". Whitespace is then replaced with special character transforming the string to "\$lorem\$ipsum\$" for $q = 2$. The string is then broken down to q-grams to produce the collection $\{\$l, lo, or, re, em, m\$, \$i, ip, ps, su, um, m\$\}$.

4.3.2 Implementation Details of the Clustering Algorithms.

To compare the clustering algorithms, we implemented the algorithms based on the description given by their authors' in the R programming language[7]. We report running times, but they should be taken as an upper bound on the computation time. Our implementations could be optimized further and, more notably for our study, we have tried to ensure the time optimization is equitable. The implementation used for the Affinity Propagation Clustering Algorithm was based on a modified version of the APCluster library [13]. We modified the library to use sparse matrices [8], instead of dense matrices. Some of the clustering algorithms were not originally designed for an input similarity graph and therefore we needed to make decisions on how to transform the similarity graph to suit the algorithm's input format. The original implementation of the Ricochet algorithms obtained from the authors worked only for complete graphs (meaning graphs containing all edges where some edge weights many be zero). In our implementation, we fixed this issue. In Correlation Clustering, we build the input correlation graph by assigning '+' to edges between nodes with similarity greater than θ, and assign '−' to edges between nodes with similarity less than θ.

For the results in this paper, we use the term Correlation Clustering to refer to the approximation algorithm CC-PIVOT [2] (CC-PIV), which is a randomized expected 3-approximation algorithm for the correlation clustering problem. As described in Section 3.3, CC-PIV is similar to the partition variant of the STAR algorithm, but the vertices are chosen randomly and not from an ordered list. Note that CC-PIV is actually classifiable as Cut-based clustering, but this is not necessarily true for all correlation clustering algorithms.

For the MCL algorithm, we employed an R implementation of MCL, based on the original C implementation.[8] As noted previously, we fix the inflation parameter of the MCL algorithm for all the datasets and treat it as an unconstrained algorithm. We use the default parameter value ($I = 2.0$) as recommended by the author of the MCL algorithm. It should be noted that the MCL algorithm and the MCL process described by van Dongen [65] are not always equivalent.[9] Misunderstandings of the difference have lead to some contradictory results in the past [15]. Although the MCL process is used in the MCL algorithm, the MCL algorithm can make use of approximations. With respect to such an implementation, a top-k matrix-element approximation is used during the expansion phase, effectively changing the complexity of the algorithm.[9] This forces faster convergence of the process for very large graphs that are not capable of being stored in available memory[9]. This approximation was not used during the experimental trials.

For the Cut Clustering algorithm, we implemented an R version of the Ford-Fulkerson algorithm. We evaluated different values for the parameter α across a subset of the datasets at varying thresholds to find the α value that would produce a total number of clusters (for each dataset) that was closest to the ground truth. We found that $\alpha = 0.2$ worked best and used this value throughout our tests. We used the R programming language [23, 46, 8, 13] for all implementations.

The different centralities measures were observed using the Star algorithm[10] and the Center algorithm. Upon closer inspection of the Center clustering algorithm, we realized it was a subroutine used by the non-sequential Ricochet algorithms. We applied centrality measures to the Center clustering algorithm in an attempt to avoid the KL-heuristic used by the Ricochet algorithms. The KL-heuristic is responsible for the runtime complexity difference between the Star and the Ricochet Clustering algorithms. Avoiding the heuristic was achieved by using centralities to remove the arbitrary labeling decisions made by CENTER (see Algorithm 1).

We also test the Mean centrality measure in setting exemplar preferences for AP. We were limited in using centralities for setting initial exemplar preferences, because prefer-

[7] Source code is available at https://github.com/hussaibi/libclustER.

[8] http://micans.org/mcl/src/mcl-06-058/

[9] http://micans.org/mcl/man/mclfaq.html

[10] Evaluation was motivated by past work which was not in the context of string duplicate detection.

ences have to be related to the notion of similarity used. Additionally, AP's parameters for deciding convergence were changed from the default. The number of iterations deciding convergence was changed to 10, and the maximum number of iterations was changed to 100. The values used were purposely lowered from the default settings because poorer accuracy measures were being returned for substantially longer run-times.

4.3.3 Limitations

In our evaluation, we focused on a θ threshold of $[0.1, 0.5]$. We have found that under this range of θ values, the accuracy measures used for our evaluations (detailed in Section 5) are less correlated (across the different clustering algorithms and the different synthetically-generated data-sets), thereby warranting further investigation. This section briefly describes our justification and evaluation process.

For each algorithm and data-set, we calculated multiple accuracy measures. These measures were compared in a pairwise manner for fixed θ-thresholds and clustering-algorithms. We used the data at all error levels to compare the relationship between measures to exclude thresholds with high multi-collinearity (i.e. thresholds where all changes in accuracy measures can be described using the changes in a single accuracy measure). We summarized the accuracy measures across the different datasets using Principle Component Analysis (PCA) to pick basis vectors for calculating a statistical summary with minimal covariance between accuracy measures. A Gaussian distribution was used to summarize the data for a given fixed θ-value and algorithm. We then evaluated the different clustering algorithms, using each of our accuracy measure summaries, for $\theta \in [0.1, 0.9]$.

For θ over 0.5, we observed that: (a) the accuracy deteriorates, because as we remove more edges, the graph becomes increasingly sparse; and (b) the accuracy measures are highly correlated. This indicates that the clusterings can be evaluated using a single accuracy measure, and we can consequently compute our other accuracy measures due to the strong correlation and covariance present. Hence, a detailed evaluation using different accuracy measures is not needed at these higher thresholds.

For θ in the range of [0.1, 0.5], we observed that while the graph is dense (an increased number of edges are available), the accuracy measures are not strongly correlated (and the covariance between accuracy measures is weak). Hence, further study is needed at these threshold values to evaluate the accuracy of the clusterings.

We omit accuracy measures for $\theta = 0.1$ as relatively high collinearity was observed at this threshold value. Specifically, a few algorithms (including the one used as the baseline for our experiments) produced clusterings consisting of only one cluster (trivial case).

5 Accuracy Measures

In this section, we explain the accuracy measures used in our evaluations. Accuracy measures evaluate the quality of clusters when compared against a ground truth.

There has been some recent work on the axiomatic approach to measure different quality aspects of clustering algorithms [1, 56, 52]. Different clustering algorithms aim to optimize different (implicit or explicit) objective functions and therefore are likely to produce different clusterings even with the same data. Thus, axiomatization of clustering-quality measures should provide a principled method to compare the pros and cons of each clustering algorithm and evaluate its significance for each application context. One important consequence of such an axiomatization effort is the natural emergence of some impossibility theorems stating that having a single clustering algorithm that can satisfy all given axioms is impossible or near impossible with some probability. Thus, some proposals to relax the original axioms have been attempted [1, 56]. This motivates our work by seeking out more measures to describe clustering accuracy. We evaluate the different clustering algorithms presented in our paper with multiple measures. It is most likely that no single clustering algorithm can outperform every other clustering algorithm for all datasets and is not capable of satisfying all desirable distance properties.

In this section, we briefly introduce what the previous axiomatic approach work entails, followed by describing a more general family of accuracy measures derived from the work by Menestrina et al. [57] and some additional accuracy measures that we introduce in our paper.

5.1 Lattice of Partitions

Meilă uses the concept of a *Lattice of Partitions* as a formal geometric representation to compare partitions [56]. The lattice is constructed by applying splitting and merging operations on a partition in order to transform it into another partition. Edges in the lattice represent the application of the merge and split operations. The edges of the lattice are undirected since splitting is the inverse operation of merging. Partitions force a mutual exclusion of all their member elements, like non-overlapping clusters.

Functions can be calculated along these lattice edges to compute an accuracy measure. Specifically, each point of the lattice represents the partitions, and edges represent splitting or merging operations. By traversing a path on the lattice, partitions are having a sequence of splitting and merging

operations applied. By associating a function to each operation, it is possible to obtain a value per operation. This provides the basis for computing a measure to compare two different partitions over one operation. Deciding how these functions are combined when computed over a path in the lattice results in a measure for comparing partitions. It also provides a method for comparing the endpoints of the lattice-path, which is precisely what an accuracy measure is doing when comparing a partitioning produced by an algorithm to the true partitioning of the elements (the ground truth).

5.1.1 An Example

Here we show why the lattice of partitions is a useful abstraction for accuracy measure construction. We will use the partitions $A = \{\{1,2,3,4\}\}$, and $B = \{\{1\},\{2\},\{3,4\}\}$ in this example. Suppose we want a distance measure, d, which indicates the number of edits required for one clustering to be transformed into another.[11] Both A and B can be interpreted as points or vertices on the lattice of partitions. Edges of the lattice represent transformations caused by splits or merging of partitions. No splits or merges are required to transform A into itself. Additionally, a single split or merge should result in one edit.

$$d(A, A) = 0 \tag{8}$$

$$d(A, \{\{1\},\{2,3,4\}\}) = 1 \tag{9}$$

$$d(\{\{1\},\{2,3,4\}\}, A) = 1 \tag{10}$$

However, we can contradict this by simply splitting an element and merging it back. Intuitively, we know this is not correct. In terms of the lattice of partitions, a circular path is valid and consistent with the restrictions we've placed so far.

$$d(A, A) = d(A, \{\{1\},\{2,3,4\}\}) + d(\{\{1\},\{2,3,4\}\}, A) \tag{11}$$

This means we need to restrict our measure in terms of what paths are valid along the lattice. The simplest solution would be to perform splits before merges. Our measure is now broken up in terms of a splitting phase followed by a merging phase.

$$d(C, D) = d_{split}(C, E) + d_{merge}(E, D) \tag{12}$$

We have removed the contradiction. This was enforced by introducing phases as part of the measure, which can be formalized as axioms of the distance measure.

$$d_{split}(A, A) = 0 \tag{13}$$

$$d_{merge}(A, A) = 0 \tag{14}$$

As a result, our measure now behaves as we intended.

$$d(A, A) = d_{split}(A, A) + d_{merge}(A, A) = 0 \tag{15}$$

There are other aspects that could also be formalized with axioms. For example, what if there are multiple valid pathways between two endpoints in the lattice of partitions? Is this acceptable due to an invariant property for picking valid paths, or would further axioms be needed to restrict the paths available?

$$A \to \{\{1\},\{2,3,4\}\} \to B \tag{16}$$

$$A \to \{\{2\},\{1,3,4\}\} \to B \tag{17}$$

These are the types of questions the axiomatic approach attempts to solve. The lattice of partitions serves as a useful abstraction when deciding on further axioms to enforce on a measure. For further information, we refer the reader to the prior work of Meilă [56, 55].

5.2 Taxonomy

The intuitive consequences of the properties arising from an axiomatic approach for accuracy measure construction were made more explicit in Meilă's later work [55], where two broad accuracy measure categorizations were highlighted, *counting pairs* and *set cardinality* based accuracy measures. Note that these classifications are not mutually exclusive. We also provide examples of measures for each categorization. The definitions for these measues can be found in Section 5.4.

5.2.1 Set Cardinality

Set cardinality accuracy measures do not align to the lattice of partitions.[12] The lattice structure is not referenced for calculating accuracy. Instead, a specific partition refinement is used for computation. These measures compare the partitions based on their content by using set operations and matches (i.e. maximal coverings). Examples of such measures in our paper include the precision, recall, and the F_1-measure (see Section 5.4). However, these measures suffer from the *partition matching problem* [55]. This problem refers to how set cardinality measures account for the accuracy of the best match of partitions, but ignore the accuracy of partitions that do not constitute the best match. In later work [22], the K-measure was proposed to address this problem by comparing all partitions, rather than using only a best match in the form of a maximal match. However, even

[11] This is an actual accuracy measure, known as basic merge distance (BMD).

[12] See Lemma 5 of Meilă's previous work for a detailed reasoning of how this can occur [56].

this measure suffers from drawbacks. Specifically, by comparing all partitions the measure prefers coarser partition refinements than almost perfect partitions with singleton partitions [1]. We refer the reader to Section 5.4 for further details and definitions of all the measures mentioned.

Note that unlike lattice-aligned accuracy measures, it is easier to extend these measures from partitioning to clustering problem domains, where elements are allowed to have membership in multiple partitions (also referred to as overlapping clusters).

5.2.2 Counting Pairs

Counting pairs measures refer to pairs of elements which appear together within a partition (as opposed to pairwise across partitions being compared). Rather than elements being members of a partition, pairs of elements are members of partitions. With respect to the measures in this paper (see Section 5.4), the Clustering Precision (CPr) , Penalized Clustering Precision (PCPr) , and Variation of Information (VI) [66, 55] accuracy measures are counting-pair accuracy measures. These measures are included in this paper because they have efficient implementations using the General-Merge-Distance [57], and were proposed specifically for evaluating duplicate detection results.

5.3 General Merge Distance

Menestrina et al. built upon the work of Meilă by formulating a General-Merge-Distance (GMD), which they categorize as an edit-based accuracy measure [57]. Unlike set-cardinality-based accuracy measures, GMD only calculates distances aligned to the lattice of partitions. This means the lattice used to transform one partition to another is referenced as input during the computation. GMD performs all split operations prior to merge operations. The defining features of GMD (what makes it an edit-based measure) is that the distance functions calculated along the lattice cannot take clusterings as input, but rather scalars corresponding to the clusterings (i.e. cluster size, probability of occurrence, etc.). Additionally, all distance functions calculated over splits and merges must be order-independent (return the same value regardless of the applied ordering of lattice operations). All of these properties allow GMD to guarantee a linear runtime complexity making it ideal for large duplicate detection tasks.

5.4 Measure Details

In order to evaluate the quality of the clusters produced by the clustering algorithms, we use several accuracy measures from the clustering literature and also measures that are suitable for the final goal of duplicate detection. Suppose that we have a set of k ground truth clusters $G = \{g_1, \ldots, g_k\}$ of base relation R. Let $C = \{c_1, \ldots, c_{k'}\}$ denote the set of k' output clusters of a clustering algorithm. We define a mapping f from the elements of G to the elements of C, such that each cluster g_i is mapped to a cluster $c_j = f(g_i)$ that has the highest percentage of common elements with g_i. Precision and recall for a cluster g_i, $1 \leq i \leq k$ is defined as.

$$Pr_i = \frac{|f(g_i) \cap g_i|}{|f(g_i)|} \tag{18}$$

$$Re_i = \frac{|f(g_i) \cap g_i|}{|g_i|} \tag{19}$$

Intuitively, the value of Pr_i is a measure of the accuracy with which cluster $f(g_i)$ reproduces cluster g_i, while the value of Re_i is a measure of the completeness with which $f(g_i)$ reproduces class g_i. Precision, Pr, and recall, Re, of the clustering are defined as the weighted averages of the precision and recall (respectively) values over all ground truth clusters. The F_1-measure is defined as the harmonic mean of precision and recall.

$$Pr = \sum_{i=1}^{k} \frac{|g_i|}{|R|} Pr_i \tag{20}$$

$$Re = \sum_{i=1}^{k} \frac{|g_i|}{|R|} Re_i \tag{21}$$

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re} \tag{22}$$

We use precision, recall and F_1-measure as indicative values of the ability of an algorithm to reconstruct the ground truth clusters . However, in our framework, the number of clusters created by the clustering algorithms is not fixed and depends on the datasets and the threshold value used in the similarity join. Therefore, we define two other measures specifically suitable for our framework. Let CPr_i be the number of pairs (of records) in each cluster c_i that are in the same ground truth cluster $g_j : c_i = f(g_j)$, i.e.,

$$CPr_i = \frac{|(t,s) \in c_i \times c_i | t \neq s \wedge \exists j, (t,s) \in g_j \times g_j|}{\binom{k'}{2}} \tag{23}$$

We define Clustering Precision, CPr, to be the average of CPr_i for all clusters of size greater than or equal to 2. The value of CPr indicates the ability of the clustering algorithm to assign records that should be grouped together to a single clustering, regardless of the number and the size of the clusters produced. Note, Clustering Precision [41] is equivalent to the pair precision accuracy measure defined by Menestrina et al [57].

Table 3 Re-expressing some accuracy measures using General Merge Distance (GMD). Note, $\perp$ is used as a place holder for a trivial clustering of singletons, $\top$ is a place holder for a trivial clustering of one cluster, and Basic Merge Distance is included for completeness of known measure implementable by GMD.

Measure	Rewrite
BMD	$GMD(C, G, f_m = 1, f_s = 1)$
Pair Pr	$1 - \frac{GMD(C,G,f_m=0,f_s=xy)}{GMD(C,\perp,f_m=0,f_s=xy)}$
Pair Re	$1 - \frac{GMD(C,G,f_m=xy,f_s=0)}{GMD(\perp,G,f_m=xy,f_s=0)}$
VI	$GMD(C, G, f_m = f_s = H(x) + H(y) - H(x + y))$
k	$BMD(C, \top) + 1$
k'	$BMD(G, \top) + 1$

In order to penalizes algorithms that create greater or fewer clusterings than the ground truth, we define Penalized Clustering Precision, $PCPr$, and compute it as CPr multiplied by the ratio of extra or missing clusters in the result, i.e.,

$$PCPr = \begin{cases} \frac{k}{k'} CPr & k < k' \\ \frac{k'}{k} CPr & k \geq k' \end{cases} \tag{24}$$

It should be noted that Clustering Precision (CPr) falls under Meilă's framework and under the GMD framework. Specifically, CPr is equivalent to the GMD-based Pair Precision for partitions. Clusterings are treated as graph cliques, and the edges of each graph clique make up the members of a clustering (rather than vertices, as is the case with Counting Pairs measures). *Edges are unused* if they occur between clique representations (inter-clique edges) for either the produced or ground-truth clusterings. In short, CPr measures how well the transitivity of the similarity-join has been preserved in the ground-truth. As a result, CPr is more sensitive than precision to displaced records with respect to the size of the cluster. CPr is less sensitive to displacements in large clusters, than it is to those in smaller clusters. Hence more importance is given to records with less duplicates. In the GMD framework, the functions f_s and f_m are the functions applied over split and merge operations (as mentioned in Section 5.3).

$$CPr = 1 - \frac{GMD(C, G, f_m = 0, f_s = xy)}{GMD(C, \perp, f_m = 0, f_s = xy)} \tag{25}$$

We include the K-measure that measures the amount of fragmentation or incompleteness between a produced clustering and a ground truth clustering. The K-measure differs from the F_1-measure by observing all matches, instead of the best matches. The K-measure is defined as the geometric mean of Average Cluster Purity (ACP) and Average Author Purity (AAP). The ACP and AAP are defined as the fragmentation and incompleteness between the produced and ground truth clusterings, respectively (see equations 26 and 27). The ACP and AAP are similar to precision and recall, respectively. ACP and AAP differentiate from precision and recall by considering all cluster mappings (as opposed to maximal mappings, f). The maximal mappings used in calculating the F_1-measure are a strict subset of the mappings used in the K-measure. As a result of these extra mappings, it is possible to create conditions for the K-measure to fluctuate while the F_1-measure remains unchanged (due to unchanged maximal mappings). This results directly from ACP (AAP) being a mean of means but precision (recall) being a mean of maximums. In terms of duplicate detection, F_1 evaluates accuracy by assuming the best matchings to be the default usage of a produced clustering for detecting duplicates. This default usage is not assumed by the K-measure. As a result, a mean of means is calculated. The K-measure assumes all matchings to be possible with respect to the default usage of the produced clusters. The K-measure's bias for coarser cluster refinements [1] stems from this assumption. The K-measure rewards a produced clustering that can be turned into a ground truth clustering exclusively with either split or merge operations. If both operations are required, then the K-measure will penalize the accuracy of the produced clustering (both ACP and AAP are contributing to the penalty).

$$ACP = \frac{1}{|V|} \sum_{c \in C} \frac{\sum^{g \in G} |c \cap g|^2}{|c|} \tag{26}$$

$$AAP = \frac{1}{|V|} \sum_{g \in G} \frac{\sum^{c \in C} |c \cap g|^2}{|g|} \tag{27}$$

$$K_{measure} = \sqrt{ACP \times AAP} \tag{28}$$

By definition, ACP and AAP do not take into account clusters which overlap. As such, we contribute a new definition for this accommodation by weighing the vertices of the graph, V. We treat overlaps as fuzzy clusterings, where reoccurring vertices have uniform partial membership to the containing clusterings [47]. We use partial membership to avoid using the notion of best match. The effect is given in equation 29, where the size of a vertex is changed to summing the inverse number of occurrences of each vertex in a clustering (see equation 30). The sum of $size$ of all vertex instances in a clustering will always equal to one. We enforce partial membership by using $size$ to calculate the size of clusters in the ACP and AAP.

$$size(v \in V) = \frac{1}{|\{c \mid c \in C, v \in c\}|} \tag{29}$$

$$|c \in C| = \sum_{v \in c} size(v) \tag{30}$$

Variation of Information (VI) treats all vertices as being homogeneous within a cluster. Similar to the K-measure, VI also uses a complete set of comparisons (mappings) between clusterings. According to Meilă, VI was identified as being ideal for describing the performance of clustering algorithms which utilize the concept of centers, centroids or

clustering representatives. The analogy between information theory and clustering algorithms being centers represent a properly encoded source. A center's associated clusters represent bounds upon where error-correction for an encoded source may occur. With respect to duplicate detection, all non-center verticies would be duplicates created by string errors such as token-swapping, or edit errors. As such, VI makes the assumption that an entity always has a single correct representative present within a cluster. Intuitively, VI indicates the amount of remaining uncertainty associated with the produced clustering when knowing the ground truth clustering and vice versa. The higher the VI measure, the more information needs to be accounted for by the produced and ground truth clusterings. Factors increasing VI include overly fragmented, encompassing or overlapping clusters. These factors contribute to a need for better information accounting regarding uncertainty between two given clusterings, in order to reproduce one another.

$$VI(X,Y) = H(X) + H(Y) - 2I(X,Y) \tag{31}$$

$$I(x \in X, y \in Y) = P(x,y) log \frac{P(x,y)}{P(x) \times P(y)} \tag{32}$$

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} I(x,y) \tag{33}$$

$$H\left(X\right) = \sum_{x \in X} -P(x) \log_2 P(x) \tag{34}$$

The original definition of VI does not accommodate for overlapping clusters. We deal with this by proposing a different set of normalized values, so a probability space is defined. This normalization is consistent with probability calculations on partitions. Thus the original definition is used for non-overlapping clusters.

$$P\left(x \in X, y \in Y\right) = \frac{|x \cap y|}{\sum_{a \in X} \sum_{b \in Y} |a \cap b|} \tag{35}$$

VI is not bounded like F_1 and K. Hence, we use a normalized version of VI.

$$VI_{norm}(X,Y) = \frac{VI(X,Y)}{H(X) + H(Y)} \tag{36}$$

Almost all accuracy measures in this section indicate perfect accuracy by returning 1, and are normalized. VI differs in that a normalized VI value of 0 indicates perfect accuracy, while 1 indicates the worst accuracy possible. A summary of the accuracy measure characteristics mentioned in this section can be found in Table 4.

Table 4 Summary of accuracy measure characteristics.

Measure	Characteristics
Precision	Measure uses best match and is based on set cardinality. It doesn't align to lattice of partitions. It measures a cluster's ability to query elements from another cluster. A singleton cluster used for querying a best match cluster will always have perfect Precision.
Recall	Measure uses best match and is based on set cardinality. It doesn't align to lattice of partitions. It measures a cluster's ability to retrieve elements from another cluster. A cluster retrieving elements from a cluster of all elements has perfect Recall.
F_1	Measure uses best match and is based on set cardinality. It doesn't align to lattice of partitions. The harmonic mean of Precision and Recall.
CPr	Measure uses best match and is based on counting pairs. It aligns to lattice of partitions for disjoint clusters. It is more sensitive than Precision. Differentiation between coarse cluster refinements from perfectly accurate clusters is more sever than that of Precision. The usage of counting pairs penalizes coarse cluster refinements.
PCPr	Measure uses best match and based on counting pairs. It aligns to lattice of partitions for disjoint clusters. It is more sensitive than CPr. Differentiation between cluster refinements with singleton clusters from perfectly accurate clusters is more sever than that of CPr. The usage of a number of clusters ratio penalizes the formation of singleton clusters.
K-measure	Measure is based on set cardinality and considers all matches (not only best match). It is bias towards coarser cluster refinements. Indicates the fragmentation and incompleteness among two clusters being compared.
VI	Measure can be calculated using either set cardinality or counting pairs. It aligns to lattice of partitions. Logarithmic scaling makes highly accurate clusters and perfect clusters to be much closer than other measures.

6 Results

We first report the results of our experiments and observations for each individual algorithm. We use the Partitioning algorithm (see Section 3.1.1), which returns the connected components in the similarity graph as clusters, as the baseline for our evaluations. We then present a comparative performance study among the algorithms, and finally, we report the running times for each algorithm.

Most of the accuracy results reported in this paper are average results over the medium error class of datasets in

Table 1 since we believe that these results are representative of algorithm behavior when using datasets with different characteristics. We show the results from other datasets whenever the trends deviate from the medium class datasets. Our results are publicly available.[13] Lastly, in the following sections we refer to the K-measure as 'K' and the normalized variation of information measure as 'VI'. In the following sections we show our experimental results. Section 6.1 shows the behavior of the different clustering algorithms over changing similarity graph thresholds. Section 6.2 shows the effects of centrality measures on CENTER and STAR, and the exemplar preferences for AP. Lastly, Section 6.3 shows the sensitivity to similarity graph threshold, and data error amount and distributions of the different clustering algorithms.

6.1 Individual Results

In this section we present the accuracy results for our experiments.

6.1.1 Single-pass Algorithms

Figure 5 shows the mean accuracy values for the single-pass algorithms over the Medium Error Group datasets (see Table 1), with the accuracy measure ranges over the Low, Medium, and High Error Group datasets. The ranges were calculated as the difference between the maximum and minimum attainable accuracy for a specific θ threshold.

The results, in Figure 5, show CENTER performs better than the Partitioning algorithm for small θ. Merge Center's (MC) accuracy is inbetween the two other algorithms. This occurs because Partitioning creates the largest clusters. For higher θ values, Partitioning obtains higher recall and lower precision. CENTER puts many similar records into different clusters resulting in lower recall, but higher precision. MC has lower precision than CENTER but higher than Partitioning.

MC's best recall is almost as high as that of Partitioning. MC performs better for medium θ values. However, CENTER generally has a smaller accuracy range. Thus it is the most reliable among the three with respect to the amount of error present in the dataset. However, for medium-range θ values, those that do not produce clusterings close to trivial clustering cases, MC generally outperforms CENTER and Partitioning. Particular attention should be given to PCPr and CPr. For CENTER, PCPr declines more than CPr for larger θ values, showing poor quality clusterings. Looking at the K-measure verifies the cause to be due to increased fragmentation. If the k-measure had not changed, than the cause would have been the displacement of records. To displace a record means to split off a record from its correct cluster and merging it to another cluster.

Note that the number of clusters in the ground truth is 500. These results show that precision, recall and F_1 measures alone cannot determine the best algorithm since they do not take into account the number of clusters generated, this justifies using the CPr and PCPr measures. Furthermore, we can observe the high degree of sensitivity of all these algorithms to the threshold value used in the similarity graph. In general, the main takeaway from these results would be that MC's accuracy is not consistent over error groups, while CENTER is not consistent over θ thresholds. Partitioning achieves the best recall for larger θ values due to forming fewer and larger clusters (see Figure 3).

6.1.2 Star Algorithm

Figure 6 includes the results for the Star algorithm (STAR).[14] The figure shows that the algorithm has relatively poor performance in terms of accuracy when a lower threshold value is used. For medium-range threshold values, STAR begins to peak in performance. This can be largely attributed to using a first-order neighbourhood, instead of a graph component search for forming clusters. This is indicated by the F1, K and PCPr trends peaking. Initially the clusters are capable of providing better recall, like Partition, but quickly transition to the same behavior as CENTER. Interestingly, the non-overlapping cluster variants of STAR do not have one optimal threshold for all accuracy measures, resulting in slightly more complex behavior as larger θ thresholds are used. Note that the non-overlapping (partition) variant of STAR achieved slightly higher accuracy for all measures, except Recall. For higher thresholds, the quality of the clustering considerably decreases. Partitioning, MC, CENTER and STAR exhibit similar behavior for higher thresholds due to fewer edges in the similarity graph. Partitioning performs slightly better in this case due to using graph components. STAR is able to perform slight better than CENTER due to using overlapping clusters. STAR's decrease in accuracy is due to the centers in its algorithm (see Algorithms 3 and 6) being traversed based on vertex degree. Using higher threshold values decreases the degree of all nodes and makes the choice of a proper cluster center harder, resulting in clusterings of lower quality. Even with an ideal threshold, STAR's accuracy is less than the accuracy of the single-pass algorithms. STAR obtains high recall accuracy initially for low thresholds, but it is it is still lower than the initial recall of CENTER.

[13] https://github.com/hussaibi/libclustER/blob/master/results/results-parser/output.csv

[14] As stated in Section 3.2.3, the degree centrality measure is used for traversing vertices in STAR.

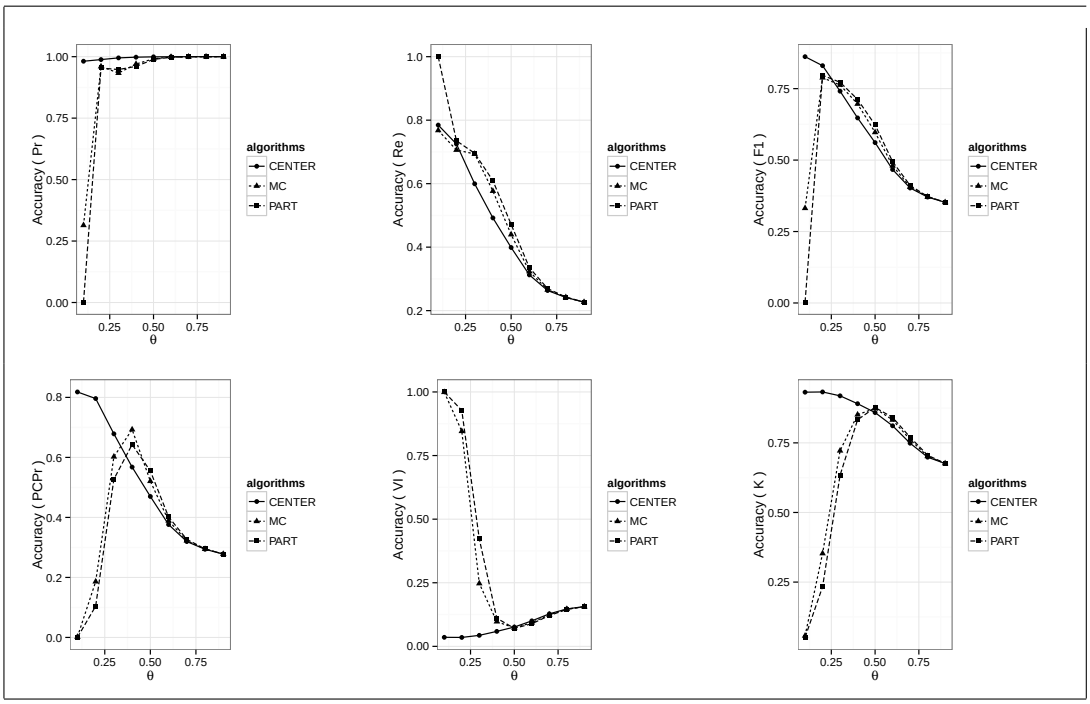
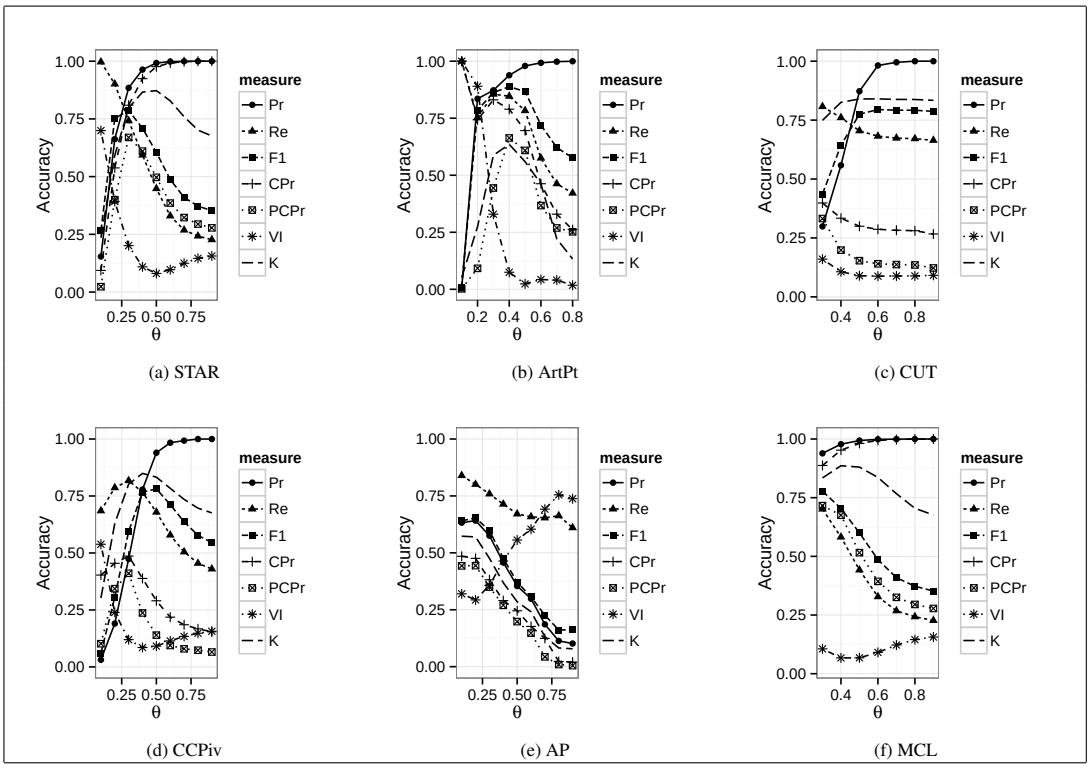


Fig. 5 Mean accuracy measures of single-pass algorithms for Medium Error Group.



(a) STAR (b) ArtPt (c) CUT

(d) CCPiv (e) AP (f) MCL

Fig. 6 Mean accuracy measures for Cut-based and Probablistic clustering algorithms over Medium Error Group.

	$\theta = 0.2$			$\theta = 0.4$		
	Part.	SR	BSR	Part.	SR	BSR
PCPr	0.101	0.628	0.466	0.645	0.590	0.578
CPr	0.991	0.821	0.868	0.879	0.754	0.895
Pr	0.104	0.989	0.675	0.788	0.991	0.828
Re	0.953	0.863	0.932	0.929	0.818	0.930
F1	0.177	0.917	0.779	0.850	0.893	0.873
Cluster#	51	735	268	704	703	323

Table 5 Sequential Ricochet Accuracy

	$\theta = 0.2$			$\theta = 0.5$		
	Part.	CR	OCR	Part.	CR	OCR
PCPr	0.101	0.494	0.351	0.469	0.402	0.687
CPr	0.991	0.967	0.981	0.805	0.782	0.817
Pr	0.104	0.434	0.299	0.934	0.958	0.862
Re	0.953	0.869	0.952	0.891	0.869	0.883
F1	0.177	0.567	0.454	0.910	0.910	0.872
Cluster#	51	258	180	994	1079	593

Table 6 Concurrent Ricochet Accuracy

6.1.3 Ricochet Algorithms

The accuracy results for SR and BSR, presented in Table 5, show that these algorithms are also more effective at lower thresholds, but are overall more robust (less sensitive) across varying threshold values. The reasoning for more effective lower thresholds can be explained via the KL-heuristic and the usage of neighbourhoods for cluster formation. For lower thresholds, non-overlapping STAR performs poorly due to fragmenting and displacing duplicate records. This can be seen by the peaking of CPr and PCPr before the peaking of the k-measure in CC-Piv (see Figure6). The initial cluster formed by STAR retains the most errors. All subsequent clusters are formed on a smaller and smaller similarity graph, making latter clusters more accurate. When similarity graph edge information is lost due to using a higher threshold, the displacement of duplicate records are less likely to occur in the initial cluster, and more likely to occur in subsequent clusters. The KL-heuristic provides the Sequential Ricochet algorithms a mechanism for dismantling the initial few clusters, by forcing record swaps with newer clusters. OCR and CR algorithms (Table 6), on the other hand, are very sensitive to the threshold value, and are more effective at higher θ values. This is again due to different way of choosing cluster seeds used in these algorithms. Marking all the nodes as seeds and gradually merging the clusters, as done in OCR and CR, results in higher quality clusters when the threshold value is high (i.e., the similarity graph is not dense) but does not work well when the threshold value is low (i.e., the similarity graph is very dense). On the other hand, when seeds are chosen sequentially based on the weight of the nodes, as done in SR and BSR, a lower threshold value (i.e., a dense similarity graph) results in more accurate weight values and therefore better choice of cluster seeds and higher quality clusters.

6.1.4 Cut Clustering

The quality of the clustering when compared to all other clustering algorithms is remarkably stable for the Cut Clustering algorithm (CUT). CUT is the most robust algorithm against both varying θ thesholds and the different error groups. Variance is largely seen in the accuracy measures CPr, PCPr, VI, and K. The relative stability of K implies the coarseness of the clusters is not changing. PCPr closely follows CPr, indicating CUT does produce relatively the same number of clusters. Upon closer manual inspection of the clusters, many singleton clusters were present in the result. The main take away would be that CUT suffers from the displacement of records based on the error group.

6.1.5 Articulation Point Clustering

The Articulation Point clustering (ArtPt) algorithm augments the Partitioning algorithm by splitting components in the graph into more refined clusters. The algorithm works best with the optimal threshold for the Partitioning algorithm (the θ value that creates partitions of highest quality in the Partitioning algorithm). ArtPt follows a similar trend to Partition, however its accuracy shows sudden improvement for the sparsest similarity graph. This sudden increase can be explained by the number of articulation points increasing for sparser similarity graphs. As such, more overlaps will occur, resulting in improvements in cluster recall. Interestingly, for large θ VI doesn't change, but other measures begin increasing. This behavior can be attributed to overlapping clusters and how they improve the completeness of clusterings, but not in a scalable manner. That is to say, K has a bias for coarser cluster refinements, but this doesn't mean larger chunks of entities haven't been clustered properly for completeness. The increase in θ increases the number of produced clusters while reducing cluster sizes, resulting in finer clusters. The steady value for VI reflects the lack of cluster fragmentation due to overlapping clusters. Since the articulation points in ArtPt are dependent on a similarity graph edges' existence, ArtPt accuracy varies across both θ and the different error groups. Although overlapping clusters should retain recall well, we observed ArtPt to perform better than STAR. This can be explained by two observations. The initial clusters of STAR are responsible for absorbing fragments. An Artpt cluster only shares articulation points with one other cluster per articulation point. As such, membership is largely non-partial, and allows for coarser clusters.

6.1.6 Correlation Clustering

CCPiv (see Figure 6) performs best when using lower to medium threshold values, producing clusters with more

complex accuracy behavior than those created by other algorithms. The quality of the produced clusters degrade at higher θ values. This is to be expected since the algorithm performs clustering based on correlation information between the nodes and a higher θ means a loss of this information. For lower thresholds, CCPiv first maximizes recall and CPr. When these two measures decline, K is observed to hit its maximum. This is a direct consequence of CCPiv's relationship to STAR. The first few clusters introduce fragmentation in the form of record displacement errors. But as the threshold rises, the amount of comparison information reduces greatly, preventing displacement from being possible to the same extent. But as more comparison information is lost for even higher thresholds, the produced clusters begin to split into more refined clusters, resulting in a steady decrease in K, CPr, and PCPr. CCPiv follows similar trends to STAR, however the quality of the clusterings is lower than STAR. This is a consequence of randomly ordering vertices, since the partition variant of STAR is similar to CCPiv and actually improves in all measure performance, with respect to the overlapping variant of STAR.

6.1.7 Markov Clustering

According to Figure 6, MCL produces clusters of increased quality than those created by the Partitioning algorithm. The MCL algorithm is most effective when used with an optimal threshold value, although it is much less sensitive overall across different error groups. This shows the effectiveness of the flow simulation process using random walks on the graph. Unlike Partitioning and CR, denser similarity graphs do not result in MCL clusters with low precision. For sparser similarity graphs, MCL drops in accuracy due to detecting duplicates with very high similarity only. This has been documented as one of the cases where MCL does not perform as well [65], where using stochastic flow becomes less meaningful. Observing CPr and PCPr, we can deduce that the number of clusters is very different between those produced and the ground truth. However, MCL also exihibits a high K, suggesting the produce clusters to be coarse cluster refinements for higher thresholds. This suggests the introduction of many vertex displacements. Upon manual inspection of the clusters, many of the clusters were singleton. From this we can infer that coarser clusters successfully capture records with high intra-cluster similarity. But with less graph edge information, boundaries for these clusters become harder to identify and were displaced into singleton clusters. Our inference is based on MCL's better retention of PCPr and CPr for higher θ values than all other algorithms.

6.1.8 Affinity Propagation Clustering

The Affinity Propagation Clustering Algorithm (AP) performs best when using lower threshold values, similar to CENTER. All AP accuracy measures show a trend of lo'sing accuracy. As the error levels increase, AP performs better for an increased range of thresholds near lower threshold values. This trend over error is not directly observable from Figure 6. Note that AP consistently improves with respect to all accuracy measures and error levels, as lower θ values are used. Lastly, AP obtains the highest recall out of all the algorithms observed. AP's recall is consistently high across all θ values.

6.2 Centrality Measures

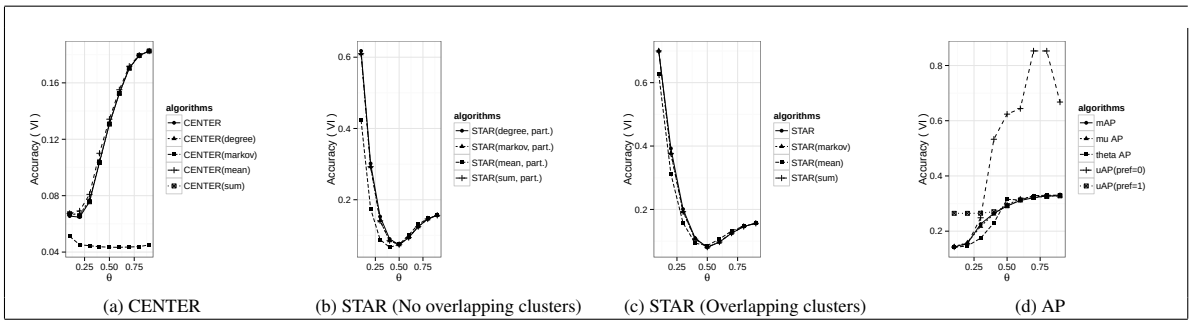
In this section we show the results of varying the type of centrality measure used for clustering (see Figure 7). We show these results by using the normalized variation of information (VI) measure, which has the property of being both a true and universal metric [56]. As such, all other quality (distance) measures will have a tendency to follow the same trends as VI for values indicating high accuracy. Note, that this doesn't mean VI is the only measure necessary for indicating accuracy. We use VI to make clear under what conditions the best accuracy was achievable. VI alone doesn't give us insight on why or how accuracy is being lost by the produced clusterings. Thus, VI is the measure of choice for accurately conveying the results for this section.

6.2.1 CENTER

The Center Clustering algorithm generally did not change across different centrality measures. The differences in performance were negligible with the exception of the Markov Steady-State centrality measure. The Markov Steady-State variant out-performed all other centrality variants of CENTER. In terms of accuracy, the Markov Steady-State variant was able to maintain a consistently low variation of information for Single Error Group data, which was accompanied with high F_1-measure and K-measure values.

6.2.2 STAR

The Star Clustering (STAR) algorithm did not significantly change performance trends when using different centralities. This held true regardless of whether overlapping clusterings were allowed. STAR was observed to obtain better performance when using the Mean centrality-measure for lower θ values. STAR achieved larger performance gains when forming non-overlapping clusters with the mean centrality measure. Note, our results are consistent with past work. [70], but we deal with a different dataset.



(a) CENTER (b) STAR (No overlapping clusters) (c) STAR (Overlapping clusters) (d) AP

Fig. 7 Mean VI of different centrality measures for STAR, CENTER, and AP. (a) Results for CENTER using the Single Error Group dataset. (b, c) Results for STAR, both the overlapping and non-overlapping cluster variants, using the Medium Error Group dataset. (d) Results for AP, using the High Error Group dataset.

6.2.3 Affinity-Propagation

The Affinity Propagation (AP) clustering algorithm exhibited many trade-offs in its behavior. Exemplar preferences set to one allowed AP to perform well for lower θ values, while zero value preference settings resulted in better performance for higher θ values, with respect to the other AP variants. The sparse-median, sparse-min, and mean centrality preference settings made AP more robust for Medium and High Error Group data. Additionally, AP converged more quickly for higher preference settings. The default preference settings of AP perform best for High Error Group data, and low θ-values. With dynamically changing preference settings (i.e., sparse minimum), AP outperforms other algorithms for these settings.[15] However, a higher value setting for exemplar preference resulted in AP becoming robust against different amounts of string errors.

6.3 Overall Comparison

In this section, we present more general impressions of the results. A summary is presented in Table 10.

6.3.1 Threshold Sensitivity

Among all the algorithms, SR, CUT, and BSR are the least sensitive to the threshold value. However their accuracy does not always outperform the other algorithms. In other algorithms, those that use the weight and degree of edges for clustering perform relatively better with lower threshold values, when the similarity graph is more dense. CENTER, STAR, CCPiv and MCL algorithms performed better with low threshold values when compared with other algorithms. The single-pass algorithms along with articulation-point clustering were generally more sensitive to the threshold value and were considerably more effective when used with the optimal threshold (where the number of components in the graph is close to the number of ground truth clusters).

[15] Specifically, MCL, MC, STAR, and Transitive-Closure.

6.3.2 Amount of Errors

The results in Table 7 show the best accuracy values obtained by the algorithms on datasets with different amounts of error, along with the difference (Diff.) between the value obtained for the High Error to Low Error Groups of datasets. Note that the accuracy numbers in this table cannot be used to directly compare the algorithms since they are based on different thresholds, and the input similarity graph is different for each algorithm. We use these results to compare the effect of the amount of error. These results suggest that the Ricochet group of algorithms, CUT and MCL algorithm are relatively more robust on datasets with different amounts of errors, i.e., they perform equally well on the three groups of datasets with the lowest drop in the cluster quality between the High Error and Low Error Group dataset.

6.3.3 Sensitivity to Error Distribution

Table 8 shows the best accuracy values obtained for the algorithms on Medium Error Group datasets with uniform and Zipfian distributions. Note that in the Zipfian dataset, there are many records with no duplicates (singleton clusters) and only a few records with many duplicates. PCPr is less indicative of the performance of the algorithms on this class of datasets. Our results show all algorithms are equally robust with respect to the distribution of errors, except for BSR and OCR which produce clusters of significantly lower quality. For the sequential algorithms, this is primarily due to the initial step of placing all vertices in one clustering. Even with the KL-heuristic, there is no mechanism in place for the representative of the initial clustering to disassociate with singletons present in the graph. As such, singletons should be clustered and removed prior to applying the algorithm, and merged with the produced cluster.

Table 7 Best accuracy values for all the algorithms over different groups of datasets

Measure	Group	Part.	CENTER	MC	Star	SR	BSR	CR	OCR	MCL	CUT	ArtPt.	CCPiv.	AP
Max.	Low	0.842	0.849	0.904	0.841	0.854	0.661	0.918	0.847	0.921	0.855	0.900	0.655	0.359
PCPr	Medium	0.645	0.638	0.695	0.614	0.633	0.578	0.718	0.687	0.768	0.689	0.680	0.410	0.513
	High	0.399	0.217	0.340	0.197	0.538	0.461	0.632	0.557	0.476	0.232	0.278	0.084	0.630
	Diff.	-0.443	-0.632	-0.565	-0.644	-0.316	-0.201	-0.286	-0.290	-0.445	-0.623	-0.621	-0.571	-0.270
Max.	Low	0.959	0.956	0.960	0.953	0.976	0.918	0.957	0.917	0.960	0.959	0.957	0.913	0.558
F1	Medium	0.910	0.887	0.918	0.892	0.920	0.873	0.910	0.872	0.921	0.913	0.907	0.781	0.712
	High	0.685	0.640	0.734	0.660	0.853	0.695	0.733	0.640	0.760	0.760	0.668	0.441	0.831
	Diff.	-0.273	-0.316	-0.225	-0.292	-0.123	-0.223	-0.223	-0.277	-0.199	-0.198	-0.288	-0.472	-0.273
Min.	Low	0.030	0.009	0.029	0.037	0.0176	0.092	0.277	0.236	0.0278	0.0202	0.001	0.033	0.380
VI	Medium	0.070	0.035	0.072	0.080	0.0411	0.428	0.259	0.236	0.067	0.03	0.01	0.085	0.245
	High	0.198	0.070	0.197	0.198	0.133	0.372	0.277	0.258	0.154	0.143	0.001	0.200	0.142
	Diff.	-0.167	-0.061	-0.168	-0.161	-0.115	-0.336	-0.018	-0.0224	-0.126	-0.123	-0.017	-0.167	-0.238
Max.	Low	0.948	0.982	0.950	0.952	0.97	0.83	0.361	0.454	0.952	0.966	0.860	0.942	0.487
K	Medium	0.877	0.932	0.873	0.872	0.93	0.457	0.406	0.459	0.886	0.948	0.635	0.849	0.635
	High	0.642	0.873	0.638	0.698	0.753	0.423	0.365	0.406	0.731	0.763	0.353	0.624	0.755
	Diff.	-0.306	-0.107	-0.311	-0.254	-0.217	-0.407	-0.045	-0.0532	-0.220	-0.203	-0.507	-0.317	-0.268
Best	Low	428	460	460	471	501	364	468	445	471	434	458	554	276
Cluster#	Medium	354	472	459	521	504	386	527	454	528	665	428	446	377
	High	919	203	200	221	470	356	643	455	236	1404	143	200	517
	Diff.	+491	-257	-260	-250	-32	-8	+175	+11	-235	+970	-315	+245	-240

Table 8 Best accuracy values for algorithms over Medium Error Group datasets with different distributions

Measure	Group	Part.	CENTER	MC	Star	SR	BSR	CR	OCR	MCL	CUT	ArtPt.	CC-PIV	AP
F1	Uniform	0.910	0.887	0.918	0.892	0.920	0.873	0.910	0.872	0.921	0.721	0.907	0.951	0.88
	Zipfian	0.936	0.936	0.938	0.934	0.873	0.463	0.935	0.697	0.937	0.819	0.934	0.91	0.873
	Diff.	+0.026	+0.049	+0.020	+0.041	-0.047	-0.411	+0.025	-0.175	+0.016	+0.098	+0.027	0.573	0.863
Cluster#	Uniform	354	472	459	521	504	386	527	454	528	665	428	555	495
	Zipfian	1018	934	1047.5	933	698.5	158	1061	992	1021	1038	1067	1516	414

6.3.4 Clusters Size Effectiveness

The results of our experiments, partly shown in Tables 7 and 8, show that none of the algorithms are capable of accurately predicting the number of clusters regardless of the characteristics of the dataset. For uniform datasets, SR algorithms perform extremely well for finding the correct number of clusters on datasets with different amounts of errors. However, this algorithm fails when it comes to datasets with a Zipfian distribution of errors. Overall, algorithms that find star-shaped clusters, namely CENTER, MC, STAR, CR and OCR algorithms, can effectively find the right number of clusters with an optimal threshold. CC-PIV and MCL also find a reasonable number of clusters at lower thresholds.

Algorithm	Mean (sec)	Stand. Dev(sec)
Partitioning	4.58e-02	1.76e-03
CENTER	3.63e-01	1.44e-01
MC	3.64e-01	1.38e-01
STAR	4.65e+00	1.34e+00
CUT	5.55e+01	6.23e+00
ArtPt.	4.83e-03	9.83e-04
CC-PIV.	3.73e+00	1.63e+00
MCL	3.78e+01	1.98e+01
AP	1.21e+03	1.59e+03
SR	1.62e+02	1.21e+02
BSR	8.46e+00	7.53e+00
CR	9.7e+01	5.95e+01
OCR	9.35e+01	5.9e+01

Table 9 Mean and standard deviation runtimes for uniform medium error level data.

6.4 Run Time and Scalability

As stated previously, in this work we focus mainly on comparing the quality of duplicates detected by each algorithm. However we do report the running times in this section, but the times taken by the different algorithms are not directly comparable, and should be taken as an upper bound on the computation time. All the implementations could be optimized further. Implementations of the algorithms (coded in the R programming language) are used to run these experiments on the same machine. Table 9 shows the running times for the algorithms run on Medium Error Group data with uniformly distributed errors described in Section 4.1, and with thresholds of 0.5 and above. All experiments were run on a computer with an Intel Core i7-2600 processor (3.4GHz), 12GB of RAM, and running the Ubuntu (12.04.1 LTS) operating system. These results support the runtime efficiency of the single-pass algorithms as well as MCL and Articulation Point clustering algorithms.

We reimplemented the clustering algorithms in the R programming language, allowing a more consistent scalability comparison. Our scalability evaluations compare runtimes (normalized by maximum runtime) over the density of the similarity-graph, $|E|/\binom{|V|}{2}$. Note, the type of plot used in Figure 8 was generated using Loess regression to smoothen trends. Smoothening allowed easy visualization of experimental results patterns in the presence of overplot-

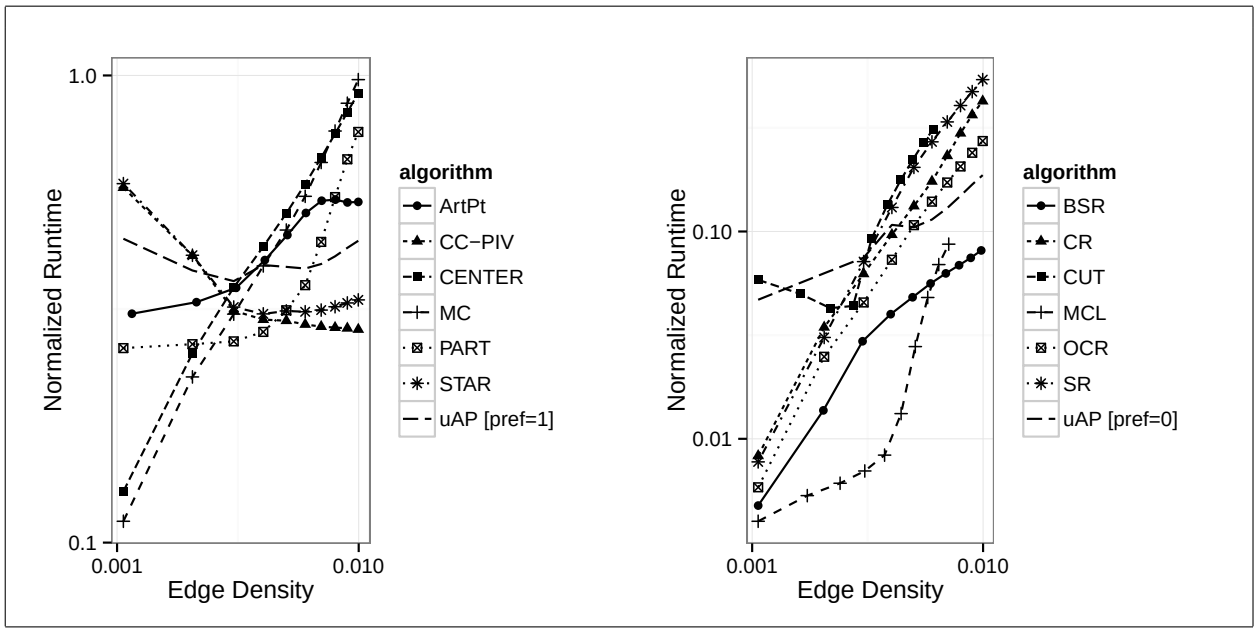


Fig. 8 Scalability Comparisons. These graphs compare normalized runtime (normalized by the max runtime for an algorithm) and edge density (of the similarity graph). A uniform trend indicates the algorithm to be scalable. The closer the entire trend is to 1 for normalized runtime, the more stable its execution behavior. Sharp growth indicates sensitivity to the size of the similarity-graph. Trend decline indicates sensitivity to other factors. These graphics use a log-log scale and show the loess regression of the normalized runtime.

ting [34]. We show the Loess regression trends of these comparisons. We found the single-pass algorithms to be the most scalable, followed by ArtPt, CCPiv, and then STAR. Although AP was found to be scalable, its runtime was not stable (for zero value preference settings), relative to other algorithms. In comparison, CUT and MCL were found to be less scalable. STAR and CCPiv have larger initial runtime because they are sensitive to both the similarity graph edge size, and the number of vertices considered. In this case, edge size becomes the determining factor of scalability for denser cases.

7 Conclusions

In this paper, we evaluated and compared several unconstrained clustering algorithms for duplicate detection by extensive experiments over various sets of string data with different characteristics. We made the results of our extensive experiments publicly available and we intend to keep the results up-to-date with state-of-the-art clustering algorithms and various synthetic and real datasets. The set of clustering algorithms previously studied [41] was expanded by including the Affinity Propagation Clustering algorithm. We include an algorithm scalability comparison, in addition to runtime comparisons. We used the K-measure and Variation of Information accuracy measures for further describing cluster quality, and the justify the accuracy measures used to describe cluster quality. We also defined and introduced a new family of clustering algorithms, which we refer to as Cut-Based clustering algorithms. Lastly, we presented the effects of centrality measures, and their use in removing arbitrary decisions in clustering algorithms. We hope these results serve as a guideline for researchers and practitioners interested in using unconstrained clustering algorithms especially for the task of duplicate detection.

Our results using the partitioning of similarity graphs (finding the transitive closure of the similarity join), which is the common approach in many early duplicate detection techniques, confirms the common wisdom that this scalable approach results in poor quality of duplicate groups. But more importantly, we show that this quality is poor even when compared to other clustering algorithms that are as efficient.

Our evaluation of centrality measures shows that algorithm behavior does significantly change for duplicate detection, by improving clustering quality. The non-overlapping variant of Star Clustering performs best using the Mean centrality measure. Center Clustering benefits the most from using the Markov Steady-State centrality measure, allowing it to better handle Single Error Group data. This variant was considered for approximating the non-sequential Ricochet algorithms, but retains the same behavior as other variants for different non-single error levels. The Affinity Propagation clustering algorithm performs well on highly erroneous data for denser similarity joins, but is outperformed in other scenarios.

The Ricochet algorithms produce high quality clusterings when used with uniformly distributed duplicates, but

failed in other distributions. The remaining algorithms were robust to different distributions. Our results also show that sophisticated but popular algorithms, like Cut clustering and Correlation clustering, gave lower accuracy than some of the more efficient single-pass algorithms. We were the first to propose the use of Affinity Propagation as an unconstrained algorithm for duplicate detection and showed that it is among the most scalable algorithms for this task.

A basic observation is that none of the clustering algorithms produce perfect clusterings. Therefore a reasonable approach is to not only keep the clustering that results from our algorithms, but to also keep the important quantitative information produced by these algorithms. In previous work [41], we show how this quantitative information can be used to provide an accurate confidence score for each duplicate that can be used in probabilistic query answering.

We do not claim that our work is completely exhaustive in terms of clustering algorithms that we cover. Specifically, considering the high popularity of some Spectral clustering algorithms, it might be worth to compare the performance of these against the unconstrained algorithms that we consider in our paper for duplicate detection.

References

1. M. Ackerman and S. Ben-David. Measures of clustering quality: A working set of axioms for clustering. In *Proceedings of Neural Information Processing Systems (NIPS)*, pages 121–128, 2008.
2. N. Ailon, M. Charikar, and A. Newman. Aggregating Inconsistent Information: Ranking and Clustering. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 684–693, 2005.
3. P. Andritsos. *Scalable Clustering of Categorical Data And Applications*. PhD thesis, University of Toronto, Toronto, Canada, September 2004.
4. A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 918–929. VLDB Endowment, 2006.
5. J. A. Aslam, E. Pelekhov, and D. Rus. The star clustering algorithm for static and dynamic information organization. *Journal of Graph Algorithms and Applications*, 8(1):95–129, 2004.
6. N. Bansal, A. Blum, and S. Chawla. Correlation Clustering. *Machine Learning*, 56(1-3):89–113, 2004.
7. N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa. Seeking Stable Clusters In The Blogosphere. In *Proceedings of the 33rd international conference on Very large data bases*, pages 806–817, Vienna, Austria, 2007. VLDB Endowment.
8. D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2010. R package version 0.999375-46.
9. R. J. Bayardo, Y. Ma, and R. Srikant. Scaling Up All Pairs Similarity Search. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 131–140, 2007.
10. O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: A Generic Approach to Entity Resolution. *VLDB J.*, 18(1):255–276, 2009.
11. I. Bhattacharya and L. Getoor. A Latent Dirichlet Model for Unsupervised Entity Resolution. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 47–58, 2006.
12. I. Bhattacharya and L. Getoor. Collective Entity Resolution in Relational Data. *Data Engineering Bulletin*, 29(2):4–12, 2006.
13. U. Bodenhofer, A. Kothmeier, and S. Hochreiter. Apcluster: an r package for affinity propagation clustering. *Bioinformatics*, 27:2463–2464, 2011.
14. S. Borgatti, K. Carley, and D. Krackhardt. On the robustness of centrality measures under conditions of imperfect data. *Social Networks*, 28(2):124–136, 2006.
15. U. Brandes, M. Gaertler, and D. Wagner. Experiments on Graph Clustering Algorithms. In *The 11th Europ. Symp. Algorithms*, pages 568–579. Springer-Verlag, 2003.
16. S. Brohee and J. van Helden. Evaluation of Clustering Algorithms for Protein-Protein Interaction Networks. *BMC Bioinformatics*, 7:488+, 2006.
17. M. Charikar, V. Guruswami, and A. Wirth. Clustering with Qualitative Information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.
18. F. Chierichetti, A. Panconesi, P. Raghavan, M. Sozio, A. Tiberi, and E. Upfal. Finding Near Neighbors Through Cluster Pruning. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '07, pages 103–112, Beijing, China, 2007.
19. W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proc. of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pages 73–78, Acapulco, Mexico, 2003.
20. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw Hill and MIT Press, 1990.
21. E. Costenbader and T. Valente. The stability of centrality measures when networks are sampled. *Social networks*, 25(4):283–307, 2003.
22. R. G. Cota, M. A. Gonçalves, and A. H. F. Laender. A Heuristic-based Hierarchical Clustering Method for

Author Name Disambiguation in Digital Libraries. In *XXII Simpósio Brasileiro de Banco de Dados*, pages 20–34, 2007.
23. G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.
24. W. H. Day and H. Edelsbrunner. Efficient Algorithms for Agglomerative Hierarchical Clustering Methods. *Journal of Classification*, 1(1):7–24, 1984.
25. E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation Clustering In General Weighted Graphs. *Theor. Comput. Sci.*, 361(2):172–187, 2006.
26. C. Demetrescu, D. Eppstein, Z. Galil, and G. F. Italiano. In M. J. Atallah and M. Blanton, editors, *Algorithms and theory of computation handbook*, chapter Dynamic graph algorithms, pages 9–9. Chapman & Hall/CRC, 2010.
27. P. Domingos and K. Kersting. Combining Logic and Probability: Languages, Algorithms, and Applications (Tutorial). In *The Twenty-Seventh AAAI Conference on Artificial Intelligence*, Seattle, USA, 2013.
28. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.
29. I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
30. P. F. Felzenszwalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *Int. J. Comput. Vision*, 59(2):167–181, 2004.
31. M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A Survey of Kernel and Spectral Methods for Clustering. *Pattern Recognition*, 41(1):176–190, 2008.
32. G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1(4):385–408, 2004.
33. L. Ford and D. Fulkerson. Maximal Flow Through a Network. *Canadian J. Math*, 8:399–404, 1956.
34. J. Fox. *Nonparametric Regression*. John Wiley & Sons, Ltd, 2005.
35. T. Frantz, M. Cataldo, and K. Carley. Robustness of centrality measures under uncertainty: Examining the role of network topology. *Computational & Mathematical Organization Theory*, 15(4):303–328, 2009.
36. B. J. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, January 2007.
37. D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 721–732. VLDB Endowment, 2005.
38. M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001.
39. O. Hassanzadeh. Benchmarking Declarative Approximate Selection Predicates. Master's thesis, University of Toronto, February 2007.
40. O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *Proc. VLDB Endow.*, 2(1):1282–1293, Aug. 2009.
41. O. Hassanzadeh and R. J. Miller. Creating Probabilistic Databases from Duplicated Data. *VLDB J.*, 18(5):1141–1166, 2009.
42. O. Hassanzadeh, M. Sadoghi, and R. J. Miller. Accuracy of Approximate String Joins Using Grams. In *Proceedings of the Fifth International Workshop on Quality in Databases*, pages 11–18, Vienna, Austria, 2007.
43. T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web. In *Proceedings of the Third International Workshop on the Web and Databases*, pages 129–134, 2000.
44. M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, 2(1):9–37, Jan. 1998.
45. B. Hussain, O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. An evaluation of clustering algorithms in duplicate detection. Technical Report CSRG-620, University of Toronto, Department of Computer Science, 2013.
46. L. J. Plotrix: a package in the red light district of r. *R-News*, 6(4):8–12, 2006.
47. A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
48. A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.
49. B. Kernighan and S. Lin. An eficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 1970.
50. A. D. King. Graph Clustering with Restricted Neighbourhood Search. Master's thesis, University of Toronto, 2004.
51. J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.
52. J. Kleinberg. An impossibility theorem for clustering. *Advances in neural information processing systems*, pages 463–470, 2003.
53. J. Kogan. *Introduction to Clustering Large and High-Dimensional Data*. Cambridge Univ. Press, 2007.
54. C. Li, B. Wang, and X. Yang. VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams. In *Proceedings of the 33rd international conference on Very large data bases*, pages 303–314, Vienna, Austria, 2007. VLDB Endow-

ment.
55. M. Meilă. Comparing clusterings—an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007.
56. M. Meilă. Comparing clusterings: an axiomatic view. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 577–584, New York, NY, USA, 2005. ACM.
57. D. Menestrina, S. E. Whang, and H. Garcia-Molina. Evaluating entity resolution results. *Proceedings of the VLDB Endowment*, 3:208–219, September 2010.
58. A. M. D. Pelleg. X-Means: Extending K-Means with Efficient Estimation of the Number of Clusters. In *Proc. of the Int'l Conf. on Machine Learning*, pages 727–734, San Francisco, CA, USA, 2000.
59. P. Perona and L. Zelnik-Manor. Self-tuning spectral clustering. *Advances in neural information processing systems*, 17:1601–1608, 2004.
60. G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.
61. S. Sarawagi and A. Kirpal. Efficient Set Joins On Similarity Predicates. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 743–754, 2004.
62. N. Slonim. *The Information Bottleneck: Theory And Applications*. PhD thesis, The Hebrew University, 2003.
63. C. Swamy. Correlation Clustering: Maximizing Agreements Via Semidefinite Programming. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 526–527, New Orleans, Louisiana, USA, 2004. Society for Industrial and Applied Mathematics.
64. C. Umans. Hardness of Approximating Sigma$_2^{\mathrm{p}}$ Minimization Problems. In *Foundations of Computer Science, 40th Annual Symposium on*, pages 465–474. IEEE, 1999.
65. S. van Dongen. *Graph Clustering By Flow Simulation*. PhD thesis, University of Utrecht, 2000.
66. N. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11:2837–2854, 2010.
67. J. Vlasblom and S. Wodak. Markov clustering versus affinity propagation for the partitioning of protein interaction graphs. *BMC bioinformatics*, 10(1):99, 2009.
68. U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.
69. J. A. Whitney. Graph Clustering With Overlap. Master's thesis, University of Toronto, 2006.
70. D. T. Wijaya and S. Bressan. Journey to the centre of the star: Various ways of finding star centers in star clustering. In *18th International Conference on Database and Expert Systems Applications*, pages 660–670, 2007.
71. D. T. Wijaya and S. Bressan. Ricochet: A Family of Unconstrained Algorithms for Graph Clustering. In *Database Systems for Advanced Applications, 14th International Conference*, pages 153–167, 2009.
72. R. Xu and I. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.
73. J. Zupan. *Clustering of Large Data Sets*. Research Studies Press, 1982.

Algorithm	Scalability	Ability to Find Correct Number of Clusters	Robustness Against		
			Chosen threshold	Error Amount	Error Distribution
Part.	H	L	L	L	H
CENTER	H	H	L	L	H
MC	H	H	L	L	H
STAR	M	H	L	L	H
SR	L	M	H	H	L
BSR	L	L	H	H	L
CR	L	H	M	H	H
OCR	L	H	M	H	L
CCPiv	H	H	L	M	H
MCL	M	H	M	M	H
CUT	L	L	L	L	H
ArtPt	H	M	L	L	H
AP	H	H	L	L	M

Table 10 Summary of the results. (H = High, M = Medium, L = Low)