Framework for Evaluating Clustering Algorithms in Duplicate Detection*

Oktie Hassanzadeh
University of Toronto
oktie@cs.toronto.edu

Fei Chiang†
University of Toronto
fchiang@cs.toronto.edu

Hyun Chul Lee
Thoora Inc.
chul.lee@thoora.com

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

The presence of duplicate records is a major data quality concern in large databases. To detect duplicates, *entity resolution* also known as *duplication detection* or *record linkage* is used as a part of the data cleaning process to identify records that potentially refer to the same real-world entity. We present the Stringer system that provides an evaluation framework for understanding what barriers remain towards the goal of truly scalable and general purpose duplication detection algorithms. In this paper, we use Stringer to evaluate the quality of the clusters (groups of potential duplicates) obtained from several unconstrained clustering algorithms used in concert with approximate join techniques. Our work is motivated by the recent significant advancements that have made approximate join algorithms highly scalable. Our extensive evaluation reveals that some clustering algorithms that have never been considered for duplicate detection, perform extremely well in terms of both accuracy and scalability.

1. INTRODUCTION

The presence of duplicates is a major concern for data quality in large databases. To detect duplicates, *entity resolution* also known as *duplication detection* or *record linkage*, is used to identify records that potentially refer to the same entity. Despite the large, and growing, number of duplicate detection techniques, the research literature comparing their quality is surprisingly sparse. There are studies and surveys comparing the similarity measures used within these techniques [16, 29, 31]. 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, their complexity, and their scalability. This diversity cries out for a study comparing the accuracy of the different clustering approaches for the duplicate detection task. In this paper, we present a thorough experimental comparison of clustering approaches from all these areas.

Our work is motivated by the recent exciting advancements that have made approximate join algorithms highly scalable [3, 7, 15, 41, 43]. These innovations lend hope to the idea that duplicate detection can be made sufficiently scalable and general purpose to be introduced as a generic, data-independent operator within a DBMS. In this paper, we describe the Stringer system[1] that provides an evaluation framework for understanding what barriers remain towards the goal of truly scalable and general purpose duplication detection algorithms. Our focus in this paper is on using Stringer to understand which clustering algorithms can be used in concert with scalable approximate join algorithms to produce duplicate detection algorithms that are robust with respect to the threshold used for the approximate join, and various data characteristics including the amount and distribution of duplicates.

1.1 Stringer Duplicate Detection Framework

In Stringer, we are interested in scalable algorithms that do not rely on a specific structure in the data. So while there are duplicate detection algorithms that can take advantage of co-citation or co-occurrence information in data such as author co-citation data or social networks, we do not consider these specialized algorithms [8, 9].[2] Our reasons are two-fold. First, such information is not always available. 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 of a social network relationship between the authors. While such social network information is common for data about people, it is less common for other types of data. Even when additional information is available, it may not be shared or may be represented differently. Therefore, in evaluating general purpose techniques, we focus on duplication detection algorithms that match two relations (on one or more attributes). Second, 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 the more specialized models that require additional structure within the data.

To ensure scalability, we consider clustering approaches which can use as input pairs of similar records that might be found by an approximate join algorithm (Figure 1). The input to the clustering is the output of the approximate join which can be modeled as a similarity graph $G(U, V)$, where a node $u \in U$ in the graph represents a record in the data and an edge $(u, v) \in V$ exists only if the two records are deemed similar. In these join techniques, two

*Supported in part by NSERC.
†Supported in part by the Walter C. Sumner Foundation.

VLDB '09, August 24-28, 2009, Lyon, France

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

[2] These techniques are sometimes called *relational*, but we avoid this term due to the obvious confusion with the relational model.

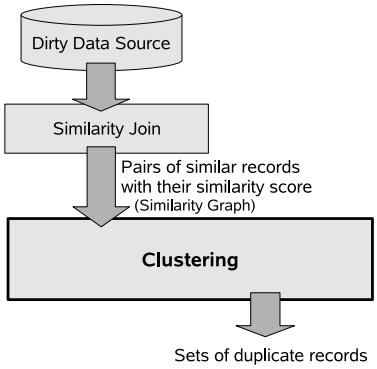

Figure 1: Duplicate detection framework

records are deemed similar if their similarity score based on a similarity function is above a specified threshold θ. The similarity graph is often weighted, i.e., each edge (u, v) has a weight $w(u, v)$ which is equal to the similarity score between the records corresponding to nodes u and v. But a key point is that these approximate join techniques are extremely proficient at finding a small and accurate set of similar items. This feature permits the effective use of clustering techniques on the output of the join, including the use of techniques that would not scale to graphs over the original input relations.

Given the result of a similarity or approximate join, a clustering algorithm can then be used to group nodes in the record similarity graph into clusters containing potential duplicates. Thus, the clustering algorithms must be *unconstrained* algorithms, that is, the algorithms do not require as input the number of clusters or other domain specific parameters. For this framework, they must be designed for graph data and should be able to produce a large (and unknown) number of clusters.

1.2 Clustering Algorithms

There exists a wealth of literature on clustering algorithms including several books [35, 40, 51], surveys [18, 23, 28, 36, 50], and theses [2, 39, 44, 47, 48]. In the classification of clustering algorithms in Figure 2 (from [35]) our algorithms fall in the partitional class. That is, we are not interested in algorithms that are supervised or only produce hierarchical clusters. As noted above, we only consider those algorithms that are *unconstrained*. The main characteristic of such algorithms is that unlike the majority of clustering algorithms, they do not require the number of clusters as input. All these algorithms share the same goal of creating clusters that maximize the intra-cluster weights, and minimize the inter-cluster edge weights. Determining the best possible set of clusters that satisfies this objective is known to be computationally intractable. Therefore several proposals have been made to find an approximate solution either based on heuristics or theoretical justifications. We consider the following clustering algorithms:

- **Single-pass algorithms** including Partitioning, CENTER and MERGE-CENTER, that efficiently perform clustering by a single scan of the list of edges of the similarity graph. Partitioning or transitive closure has been used previously as a part of many duplicate detection algorithms [33]. The CENTER algorithm has shown to be effective in web document retrieval [32]. MERGE-CENTER is a new extension of CENTER we propose to enhance the accuracy of the algorithm for duplicate detection without losing efficiency.

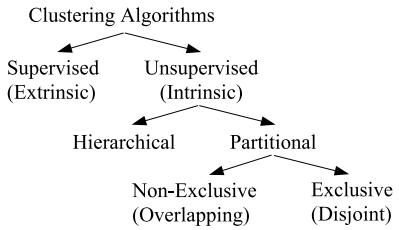

Figure 2: A classification of clustering algorithms

- **Star clustering algorithm** originally proposed for clustering documents [4], creates star-shaped clusters from the similarity graph.
- **Ricochet family of algorithms** were recently proposed as unconstrained graph clustering algorithms for document clustering [49]. These algorithms are based on combining ideas from the classic K-means algorithm and the Star algorithm.
- **Cut Clustering** algorithm based on finding minimum cuts of edges in the similarity graph, and evaluated on bibliographic citation and web data [24].
- **Articulation Point Clustering** a scalable graph partitioning algorithm based on finding articulation points and biconnected components [17], and evaluated on blog data for identifying chatter in the blogosphere [6].
- **Markov Clustering (MCL)** is a fast and scalable unsupervised clustering algorithm based on simulation of stochastic flow in graphs [47]. It has shown high efficiency and quality in applications in bio-informatics [11, 38].
- **Correlation Clustering** was originally proposed for clustering graphs with binary edge labels indicating correlation or lack of correlation of connected nodes [5]. The labels can be assigned to edges based on the similarity scores of the records (edge weights) and a threshold value, which makes it appealing as an unconstrained algorithm for clustering similarity graphs.

In this work, we do not consider algorithms that contain parameters adjusting for characteristics in the data or cluster properties (such as the cluster diameter) [13, 34]. We do consider two algorithms that contain input parameters, namely, Markov Clustering (MCL) and Cut Clustering. MCL contains a parameter that can influence coarseness and the quality of the clustering. However, as reported by previous work [49] (and also supported by our experiments) an optimal value for the parameter is rather stable across applications. The Cut Clustering algorithm has a single parameter, which as our experiments show, directly influences the quality of the clustering. We therefore include this algorithm only for completeness. We however fix the value of the parameters of these algorithms in our experiments and treat them as unconstrained algorithms.

To the best of our knowledge, clustering algorithms not considered in this paper do not meet the requirements of our framework. Specifically, the X-means algorithm [42] (an extension of K-means), that does not require as input the number of clusters, as well as popular Spectral algorithms [14, 37] perform clustering on a set of points in Euclidean space, and need the coordinates of the data points as input. Moreover, the main application of these algorithms is in pattern recognition and image segmentation, 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 [22, 26] inapplicable.

1.3 Evaluation Framework

To compare the quality of the detected duplicates, we have chosen a number of robust metrics. Part of our comparison is based on widely-used quality measures in information retrieval, namely precision, recall and the F1 measure which have previously been used for evaluation of clustering algorithms. We also use quality measures suitable for the task of duplicate detection. An important characteristic of the algorithms that should be evaluated in our framework is their ability to find the correct number of clusters.

For a thorough evaluation of the clustering algorithms, it is essential to have datasets of varying sizes, error types and distributions, and for which the ground truth is known. For the experiments in this paper, we use datasets generated by a publicly available version of the widely used UIS database generator which has been effectively used in the past to evaluate different approximate selection and join predicates used within duplicate detection [29, 31]. We follow the best practice guidelines from information retrieval and data management, to generate realistic errors in string data. Specifically, we use 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 are put in one cluster (which we use as ground truth) in order to be able to measure quality (precision and recall) of the clustering algorithms.

Although some of the algorithms considered in this paper have been evaluated previously on synthetic randomly generated graphs [10, 47], in document clustering [49], and in computational biology [11], our work is the first to compare all these algorithms for duplicate detection, and based on several robust quality measures.

1.4 Contributions and Organization

Our contributions include the following:

- We present a set of unconstrained clustering algorithms and show how they can be used, in conjunction with scalable approximate join algorithms, for duplicate detection. We include highly scalable (single pass) algorithms together with some more sophisticated and newer clustering algorithms (Correlation Clustering and Markov Clustering) 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, including the Star clustering algorithm and the Ricochet family of algorithms that were originally proposed for document clustering, as well as the graph-theoretic algorithms Cut Clustering and Articulation Point Clustering. The majority of the clustering algorithms presented in this paper were not previously used for duplicate detection.

- We present a comprehensive evaluation framework to study the behaviour of unconstrained clustering algorithms for the task of duplicate detection. This framework permits scalability by ensuring that grouping decisions can be made on the (relatively small) output of an approximate join, rather than on the original relation. Our evaluation is also based on several robust quality measures and on various datasets with many different characteristics.

- We present the results of our comprehensive evaluation and comparison study of the effectiveness of unconstrained clustering algorithms for duplicate detection over string data. Our results show the effect of the characteristics of the datasets and the similarity threshold on the accuracy of the algorithms. Specifically, we show that all algorithms with the exception of Ricochet family of algorithms are relatively robust to the distribution of the errors, although Ricochet algorithms are less sensitive to the value of the threshold and the amount of error in the datasets. We show that sophisticated but popular algorithms like Cut clustering and Correlation clusterings have lower accuracy than the more efficient single-pass algorithms. We are the first to propose using Markov clustering for duplicate detection and show that in fact it is among the most accurate algorithms for this task and is also very efficient.

This paper is organized as follows. We present an overview and brief description of the unconstrained clustering algorithms in the next section. In Section 3, we discuss the methodology as well as the results of our extensive experiments over several datasets of string data. Section 4 presents a summary of our evaluations and concludes the paper.

2. UNCONSTRAINED CLUSTERING

Unconstrained clustering algorithms aim to create clusters containing similar records $\mathcal{C} = \{c_1, \ldots, c_k\}$ where the value of k is unknown. The clustering may be exclusive (disjoint), whereby the base relation is partitioned and there is no overlap of nodes among the clusters, that is, $\bigcup_{c_i \in \mathcal{C}} c_i = R$ and $c_i \cap c_j = \emptyset$ for all $c_i, c_j \in \mathcal{C}$. Alternatively, non-exclusive clustering permits nodes 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.

Consider the source relation as a graph $G(U, V)$ where each node $u \in U$ represents a record in the base relation and each edge $(u, v) \in V$ connects two nodes u and v only if they are similar, i.e., their similarity score based on some similarity function $sim()$ is above a specified threshold θ.

2.1 Single-pass Algorithms

In this class of algorithms, we do not materialize the similarity graph. In fact, all the algorithms can be efficiently implemented by a single scan of the list of similar pairs returned by the similarity join module, although some require the list to be sorted by similarity score. We only use the graph G to illustrate these techniques. Figure 3 (from [30]) illustrates the result of applying these algorithms to a sample similarity graph.

2.1.1 Partitioning (Transitive Closure)

The Partitioning algorithm clusters the given records by finding the connected components in the graph, and returning each connected component as a cluster. The algorithm performs clustering by first assigning each node to its own cluster. Then, the list of similar pairs (the output of the similarity join) is scanned once and if two connected nodes are not in the same cluster, their clusters are merged. Figure 3(a) shows how this algorithm clusters a sample graph. As shown in this figure, the algorithm may result in big clusters, the results in many records that are not similar being put in the same cluster. Partitioning is the common approach used in early entity resolution work, and is included as a baseline.

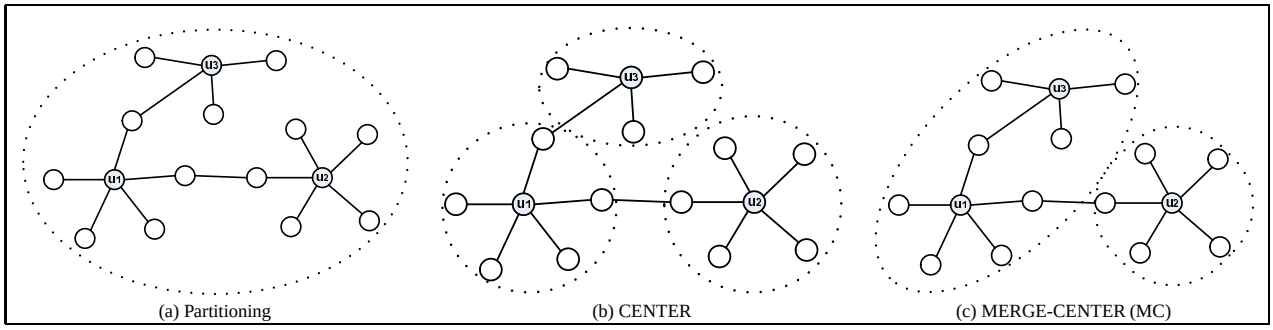

Figure 3: Illustration of single-pass clustering algorithms

2.1.2 CENTER

The CENTER algorithm [32] performs clustering by partitioning the similarity graph into clusters that have a *center*, and all records in each cluster are similar to the center of the cluster. This algorithm requires the list of the similar pairs (the output of the similarity join) to be sorted by decreasing order of similarity scores. The algorithm then performs clustering by a single scan of the sorted list. The first time a node u is in a scanned pair, it is assigned as the center of the cluster. All the subsequent nodes v that are similar to u (i.e., appear in a pair (u, v) in the list) are assigned to the cluster of u and are not considered again. 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.

2.1.3 MERGE-CENTER

The MERGE-CENTER algorithm [30] is a simple extension of the CENTER algorithm. It 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 the cluster c_i is similar to the center of c_j. This is done similarly by a single scan of the list of the similar records, 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) in the scan and are not assigned to any cluster, are assigned to the cluster of u, and are not assigned as the center of any other cluster. 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 clustering of the sample similarity graph by this algorithm, assumimg 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 of a cluster. As this figure shows, MERGE-CENTER creates fewer clusters for the sample graph than the CENTER algorithm, but more than the Partitioning algorithm.

2.2 Star and Ricochet Algorithms

2.2.1 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 [46]. The Star clustering algorithm [4] is proposed as a way to cover the graph by *dense star-shaped subgraphs* instead. Aslam et al. [4] prove several interesting accuracy and efficiency properties, and evaluate the algorithm for document clustering in information retrieval. The Star algorithm performs clustering on a weighted similarity graph $G(U, V)$ as follows:

a. Let each vertex in G be initially unmarked.

b. Calculate the degree of each vertex $u \in U$.

c. Let the highest degree unmarked vertex be a star center, and construct a cluster from the center and its associated vertices. Mark each node in the newly constructed star.

d. Repeat step c until all the nodes are marked.

Note that this algorithm is similar to the single-pass algorithm CENTER, but may produce overlapping (non-disjoint) clusters. Implementing this algorithms requires another scan of the input list of similar records to calculate the degree of each vertex and sort the vertices based on their degrees.

2.2.2 Ricochet family of algorithms

Wijaya and Bressan [49] recently proposed a family of unconstrained algorithms called 'Ricochet' due to their strategy resembling the rippling of stones thrown in a pond. 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 (similar to the Star algorithm). In all four algorithms, a weight is associated with each vertex which is equal to the average weight of their adjacent edges. We briefly describe the four algorithms below and refer the reader to [49] for the complete algorithm pseudo-code.

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 the seed of their current cluster. If there are no re-assignments, then no new cluster is created. If a cluster is reduced to singleton, it is reassigned to its nearest cluster. The algorithm stops when all nodes are considered.

Balanced Sequential Rippling (BSR) is similar to the sequential rippling in 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.

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 seed, the vertex is assigned to the cluster of the seed. If the vertex is a seed, it is assigned to the cluster of the other seed only if its weight is smaller than the weight of the seed. This iteration (propagation of ripple) is performed at equal speed for all seeds. This requires sorting of 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.

Ordered Concurrent Rippling (OCR) performs clustering similar to concurrent rippling but removes the requirement that the rippling propagates at equal speeds. Therefore this algorithm is relatively more efficient and also could possibly create higher quality clusters by favoring heavy seeds.

2.3 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 [5], 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[5]. Thus, several attempts have been made to approximate both the maximization and minimization formulations [5, 12, 19, 45]. 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 purpose of our application, we implemented and evaluated the algorithm Cautius in [5]. 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.

2.4 Markov Clustering (MCL)

The Markov Cluster Algorithm (MCL), proposed by Stijn van Dongen [47], 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 are finally ended when we find regions (clusters) with strong internal flow that are separated by boundaries with hardly any flow.

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, and the algorithm can be adapted to different contexts.

2.5 Cut Clustering

Given a directed graph $G = (U, V)$ 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.[3] 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 [25] 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 partition of the vertices into two sets, where s and t are in different sets, such that the cut weight (the sum of edge weights in the cut) 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 [24]. 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.

There have been many algorithms proposed for finding the minimum cut of G, including finding augmenting paths by Ford and

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

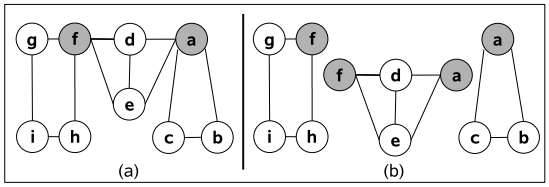

Figure 4: (a) Articulation points are shaded, (b) Biconnected components.

Fulkerson [25], Edmonds and Karp [21], and Dinic [20], and other variations suited for dense graphs or sparse graphs. In our implementation, we use the push-relabel algorithm for finding minimum cuts which has been shown to perform as well as the best techniques for sparse and dense graphs [27]. 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 3.3.

2.6 Articulation Point Clustering

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. 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 [17]. The 'removal'[4] of all articulation points separates the graph into biconnected components. These components are returned as clusters of G. 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 [6] for details of a scalable and memory efficient implementation of the algorithm and its pseudo-code.

3. EXPERIMENTAL EVALUATION

In this section we describe our evaluation methodology and present an overview of the results of our experiments. We first briefly discuss the characteristics of the datasets used in the experiments. We then explain the settings of our experiments including the similarity measure used for the approximate join, and finally present results of our extensive experiments.

3.1 Datasets

The datasets used in our experiments are generated 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 [29, 33]. We use 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 are put in a single cluster (which we use as 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 (character insertion, deletion, replacement or swap)[5] token swap errors and domain specific *abbreviation errors*, e.g., replacing `Inc.` with `Incorporated` and vice versa. We use two different clean sources of data: a data set consisting of *company names* that contains $2,139$ records (name 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 average 33.55 characters record length and average 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 create data sets of up to $100K$ records.

For the results in this paper, we used 29 different datasets (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). Note that we limited the size of the datasets for our experiments on accuracy due to the fact that some of the algorithms do not scale well. However, we run experiments on much larger datasets with the algorithms that do scale and the trends were similar. Following [29], we believe the errors in these datasets are highly representative of common types of errors in databases with string attributes.

3.2 Accuracy Measures

In order to evaluate the quality of the duplicate clusters found 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 the 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 follows:

$$Pr_i = \frac{|f(g_i) \cap g_i|}{|f(g_i)|} \quad \text{and} \quad Re_i = \frac{|f(g_i) \cap g_i|}{|g_i|}$$

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 values over all

[4]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.

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

[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

ground truth clusters. More precisely:

$$Pr = \sum_{i=1}^{k} \frac{|g_i|}{|R|} Pr_i \quad \text{and} \quad Re = \sum_{i=1}^{k} \frac{|g_i|}{|R|} Re_i$$

F_1-measure is defined as the harmonic mean of precision and recall, i.e.,

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

We use precision, recall and F_1-measure as indicative values of the ability of an algorithm to reconstruct the indicated clusters in the dataset. 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.,

$$CPr_i = \frac{|(t,s) \in c_i \times c_i | t \neq s \wedge \exists j \in 1 \ldots k, (t,s) \in g_j \times g_j|}{\binom{k'}{2}}$$

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 in the same cluster to a single cluster, regardless of the number and the size of the clusters produced. In order to penalizes those algorithms that create greater or fewer clusters than the ground truth, we define Penalized Clustering Precision, $PCPr$, and compute it as CPr multiplied by the percentage of extra or missing clusters in the result, i.e.,

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

3.3 Settings

Similarity Function There are a large number of similarity measures for string data that can be used in the similarity join. Based on the comparison of several such measures in [31], 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. Weighted Jaccard similarity is the weighted version of Jaccard similarity, i.e.,

$$sim_{WJaccard}(r_1, r_2) = \frac{\sum_{t \in \mathbf{r}_1 \cap \mathbf{r}_2} w(t,R)}{\sum_{t \in \mathbf{r}_1 \cup \mathbf{r}_2} w(t,R)} \quad (1)$$

where $w(t,R)$ is a weight function that reflects the commonality of the token t in the relation R. We choose the commonly-used Inverse Document Frequency (IDF) weights, with a slight modification based on the RSJ (Robertson/Sparck Jones) weights which was shown to make the weight values more effective [29]:

$$w(t,R) \quad = \quad log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right) \quad (2)$$

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

Note that this similarity predicate can be implemented declaratively and used as a join predicate in a standard RDBMS engine [29], or used with some of the specialized, high performance, state-of-the-art approximate join algorithms [3, 7]. In our experiments we used q-grams of size 2 and the q-gram generation technique proposed by [29]: strings are first padded with whitespaces at the beginning and the end, then all whitespaces are replaced with $q-1$ special symbols (e.g., \$).

Implementation Details of the Clustering Algorithms. To compare the clustering algorithms, we have either implemented or obtained an implementation of the algorithms from their authors. Notably, not all of these algorithms have previously been implemented nor evaluated (even on their own), so we created some new implementations. For other algorithms where the authors provided us with an implementation, their implementation may have been in a different language and used different data structures from our own implementation. As a result, the time taken by different algorithms is not directly comparable. We report running times, but they should be taken as an upper bound on the computation time. All the implementations (our own and those of others) could be optimized further and, more notably for our study, we have not tried to ensure the time optimization is equitable. Rather, we are focusing on comparing the quality of the duplicates detected by each approach.

Some of the clustering algorithms were not originally designed for an input similarity graph and therefore we needed to make de-

cisions 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. Therefore, for all pairs of disconnected nodes (their similarity score was below θ), we added an edge with a small constant weight. In our implementation, for the SR and BSR algorithms we used a constant value of 0.05, and for the CR and OCR algorithms we used a constant value of 0. 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 Cautious [5] for general weighted graphs. A more efficient approximation is the CC-PIVOT algorithm [1], which is a randomized expected 3-approximation algorithm for the correlation clustering problem. This algorithm is similar to the CENTER algorithm, but the center nodes are chosen randomly and not from the sorted list of the edges. Based on our experiments, this randomization did not improve the quality of the clusters on average comparing with the CENTER algorithm, and we therefore do not include the results in this paper.

For the MCL algorithm, we employed the latest C implementation of MCL, provided by the original author of the algorithm.[7] 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 algorithm. For the Cut Clustering algorithm, we used a C implementation of the push-relabel algorithm available from the authors.[8] 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.

3.4 Results

We first report the results of our experiments and observations for each individual algorithm. We use the Partitioning algorithm, 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 behaviour when using datasets with different characteristics. We show the results from other datasets whenever the trends deviate from the medium class datasets. Our extensive experimental results over all the 29 datasets in Table 1 are publicly available at:
http://dblab.cs.toronto.edu/project/stringer/clustering/

3.4.1 Individual Results

Single-pass Algorithms. The table below shows the accuracy values for the single-pass algorithms over medium-error datasets and two thresholds that result in the best average F1 measure and the best average PCPr values in these algorithms. Similar trends were observed for the other thresholds and datasets.

[7] http://micans.org/mcl/src/mcl-06-058/
[8] http://www.avglab.com/andrew/soft.html.

	Partitioning		CENTER		MC	
	Best PCPr	Best F_1	Best PCPr	Best F_1	Best PCPr	Best F_1
PCPr	0.554	0.469	0.638	0.298	0.695	0.437
CPr	0.946	0.805	0.818	0.692	0.940	0.795
Pr	0.503	0.934	0.799	0.971	0.658	0.958
Re	0.906	0.891	0.860	0.805	0.950	0.885
F1	0.622	0.910	0.825	0.877	0.776	0.918
Cluster#	354	994	697	1305	459	1030

The results above show that, assuming that the optimal threshold for the similarity join is known for, the CENTER algorithm performs better than the Partitioning algorithm, and that the MC algorithm is more effective than both CENTER and Partitioning over these datasets. This is to be expected since Partitioning puts many unsimilar records in the same cluster resulting in higher recall, but considerably lower precision. CENTER puts many similar records in different clusters resulting in lower recall, but higher precision. These results show that MC creates clusters with precision lower than CENTER but higher than Partitioning, with a recall that is almost as high as that of Partitioning.

Note that the number of clusters in the ground truth is 500. The last row in the table shows the number of clusters generated by each algorithm. 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 join, although MC is less sensitive to this value among the single-pass algorithms.

Star algorithm. The table below gives the results for the Star algorithm, revealing that the algorithm has a better performance in terms of accuracy when a lower threshold value is used. With a lower threshold value, the Star algorithm significantly outperforms the Partitioning algorithm, and is less sensitive to the value of the threshold used. However, for higher thresholds, the quality of the clustering considerably decreases. This is because the star centers in this algorithm are chosen based on the degree of the nodes. Using higher threshold values decreases the degree of all the nodes in the graph and makes the choice of a proper cluster center harder, which results in clusters of lower quality. It is also worth mentioning that even with an ideal threshold, the Star algorithm's accuracy is less than the accuracy of the single-pass algorithms.

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	Star	Part.	Star	Part.	Star
PCPr	0.101	0.614	0.554	0.601	0.645	0.445
CPr	0.991	0.726	0.946	0.801	0.879	0.781
Pr	0.104	0.588	0.503	0.778	0.788	0.900
Re	0.953	0.730	0.906	0.842	0.929	0.870
F1	0.177	0.644	0.622	0.805	0.850	0.884
Cluster#	51	521	354	715	704	949

Ricochet family of algorithms. The accuracy results for SR and BSR, presented in the table below for two similarity threshold values, show that these algorithms are also more effective at lower thresholds, but are overall more robust (less sensitive) across varying threshold values.

	$\theta = 0.2$			$\theta = 0.4$		
	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

OCR and CR algorithms, on the other hand, are very sensitive to the threshold value, and are more effective at higher θ values

as shown in the table below. This is again due to different way of choosing cluster seeds (or centers) 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 the 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.

	$\theta = 0.2$			$\theta = 0.5$		
	Part.	CR	OCR	Part.	CR	OC
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

Cut Clustering (MinCut). Clustering the similarity graph based on minimum cuts improves the quality of the clustering when compared to the Partitioning algorithm, as shown in the table below. This improvement is significant as we increase the threshold up to 0.4. For $\theta > 0.4$, MinCut Clustering produces the same clusters as the Partitioning algorithm, since as the input graph becomes less dense, only significantly related records remain connected and further cutting the edges does not improve the quality of the clusters.

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	MinCut	Part.	MinCut	Part.	MinCut
PCPr	0.101	0.105	0.554	0.683	0.645	0.689
CPr	0.991	0.509	0.946	0.891	0.879	0.875
Pr	0.104	0.833	0.503	0.672	0.788	0.827
Re	0.953	0.564	0.906	0.908	0.929	0.926
F1	0.177	0.671	0.622	0.771	0.850	0.873
Clstr#	51	2450	354	665	704	735

Articulation Point Clustering (ArtPt). As the results show in the following table, the Articulation Point clustering slightly enhances the Partitioning algorithm by splitting some of the components in the graph into a few more clusters. This makes the algorithm only slightly less sensitive to the threshold value. 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).

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	ArtPt.	Part.	ArtPt.	Part.	ArtPt.
PCPr	0.101	0.169	0.554	0.655	0.645	0.680
CPr	0.991	0.988	0.946	0.941	0.879	0.871
Pr	0.104	0.157	0.503	0.581	0.788	0.825
Re	0.953	0.920	0.906	0.891	0.929	0.925
F1	0.177	0.251	0.622	0.693	0.850	0.871
Cluster#	51	86	354	428	704	754

Markov Clustering (MCL). As shown in the table below, the MCL algorithm produces clusters of increased quality than those created by the Partitioning algorithm. The MCL algorithm is also most effective when used with the optimal threshold value, although it is much less sensitive overall across varying θ values. This shows the effectiveness of the flow simulation process using random walks on the graph, and that unlike the Star and SR algorithms, pruning the edges with low weights does not affect the quality of the clusters significantly, and unlike Partitioning and CR, a dense similarity graph (i.e., not pruning the low-weight edges) does not result in clusters with low precision.

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	MCL	Part.	MCL	Part.	MCL
PCPr	0.101	0.599	0.554	0.768	0.645	0.644
CPr	0.991	0.934	0.946	0.921	0.879	0.866
Pr	0.104	0.571	0.503	0.754	0.788	0.888
Re	0.953	0.951	0.906	0.952	0.929	0.925
F1	0.177	0.712	0.622	0.841	0.850	0.906
Cluster#	51	323	354	528	704	777

Correlation Clustering (CCL) This algorithm performs best when using lower threshold values, producing clusters with much higher quality than those created by the Partitioning algorithm. 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.

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	CCL	Part.	CCL	Part.	CCL
PCPr	0.101	0.612	0.554	0.542	0.645	0.406
CPr	0.991	0.711	0.946	0.762	0.879	0.748
Pr	0.104	0.596	0.503	0.803	0.788	0.914
Re	0.953	0.750	0.906	0.822	0.929	0.844
F1	0.177	0.659	0.622	0.808	0.850	0.876
Cluster#	51	538	354	753	704	1000

3.4.2 Overall Comparison

Sensitivity to the value of the threshold. Table 3 shows the accuracy results of all the algorithms for different thresholds over the medium-error class of datasets. These results can be used to compare different algorithms when using a fixed threshold, i.e., with the same similarity graph as input. Among all the algorithms, SR and BSR are 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 the edges for the clustering perform relatively better using lower threshold vlues, when the similarity graph is more dense. Therefore, CENTER, Star, Correlation Clustering and MCL algorithms perform better with low threshold values comparing with the other algorithms. The single-pass algorithms along with Articulation Point and MinCut algorithms are generally more sensitive to the threshold value and are considerably more effective when used with the optimal threshold (when the number of components in the graph is close to the number of ground truth clusters), with MERGE-CENTER being the least sensitive among them.

Effect of the amount of errors. The results in Table 4 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 and 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 therefore the input similarity graph is different for each algorithm. We use these results to compare the effect of the amount of error on different algorithms. These results suggest that the Ricochet group of algorithms 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 lowest drop in the quality of the clusters on high error datasets comparing with low error groups of datasets.

Sensitivity to the distribution of the errors. Table 5 shows the best accuracy values obtained for the algorithms on medium-error datasets with uniform and Zipfian distributions. Note that in Zipfian datasets, there are many records with no duplicates (singleton clusters) and only a few records with many duplicates. Due to the fact that the PCPr measure is the average CPr value for all the clusters and is calculated for clusters of size 2 or more, this accuracy measure is less indicative of the performance of the algorithms on this class of datasets. These results show that all the algorithms are

Table 3: Average accuracy of all the algorithms for different thresholds over medium-error datasets

θ	Measure	Part.	CENTER	MergeC	Star	SR	BSR	CR	OCR	CCL	MCL	MinCut	ArtPt.
0.2	PCPr	0.101	0.593	0.257	0.614	0.628	0.466	0.494	0.351	0.612	0.599	0.105	0.169
	F1	0.177	0.666	0.389	0.644	*0.917*	0.779	0.567	0.454	0.659	0.712	0.671	0.251
	Cluster#	51	472	134	521	735	268	258	180	538	323	2450	86
0.3	PCPr	0.554	0.638	0.695	0.601	0.616	0.564	0.718	0.578	0.542	*0.768*	0.683	0.655
	F1	0.622	0.825	0.776	0.805	0.907	0.863	0.791	0.718	0.808	0.841	0.771	0.693
	Cluster#	354	697	459	715	721	315	527	306	753	528	665	428
0.4	PCPr	0.645	0.445	0.692	0.445	0.590	0.578	0.640	0.629	0.406	0.644	0.689	0.680
	F1	0.850	0.887	0.894	0.884	0.893	0.873	0.887	0.819	0.876	0.906	0.873	0.871
	Cluster#	704	956	750	949	703	323	786	454	1000	777	735	754
0.6	PCPr	0.284	0.225	0.275	0.234	0.546	0.527	0.264	0.546	0.000	0.273	0.284	0.264
	F1	0.897	0.841	0.892	0.861	0.847	0.839	0.886	0.851	0.183	0.892	0.897	0.887
	Cluster#	1345	1650	1378	1593	634	323	1435	746	4979	1382	1345	1438
0.8	PCPr	0.175	0.173	0.174	0.173	0.604	0.458	0.174	0.340	0.000	0.175	0.175	0.174
	F1	0.773	0.757	0.768	0.761	0.768	0.730	0.768	0.730	0.183	0.772	0.773	0.765
	Cluster#	2173	2232	2188	2227	*504*	307	2196	1709	4979	2176	2173	2209

Table 4: Best accuracy values for all the algorithms over different groups of datasets

Measure	Group	Part.	CENTER	MergeC	Star	SR	BSR	CR	OCR	CCL	MCL	MinCut	ArtPt.
Max.	Low	0.842	0.849	0.904	0.841	0.854	0.661	0.918	0.847	0.818	0.921	0.855	0.900
PCPr	Medium	0.645	0.638	0.695	0.614	0.633	0.578	0.718	0.687	0.612	0.768	0.689	0.680
	High	0.399	0.217	0.340	0.197	0.538	0.461	0.632	0.557	0.175	0.476	0.232	0.278
	Diff.	-0.443	-0.632	-0.565	-0.644	-0.316	-0.201	-0.286	-0.290	-0.642	-0.445	-0.623	-0.621
Max.	Low	0.959	0.956	0.960	0.953	0.976	0.918	0.957	0.917	0.951	0.960	0.959	0.957
F1	Medium	0.910	0.887	0.918	0.892	0.920	0.873	0.910	0.872	0.876	0.921	0.913	0.907
	High	0.685	0.640	0.734	0.660	0.853	0.695	0.733	0.640	0.624	0.760	0.760	0.668
	Diff.	-0.273	-0.316	-0.225	-0.292	-0.123	-0.223	-0.223	-0.277	-0.327	-0.199	-0.198	-0.288
Best	Low	428	460	460	471	501	364	468	445	489	471	434	458
Cluster#	Medium	354	472	459	521	504	386	527	454	538	528	665	428
	High	919	203	200	221	470	356	643	455	236	236	1404	143
	Diff.	+491	-257	-260	-250	-32	-8	+175	+11	-254	-235	+970	-315

equally robust with respect to the distribution of the errors in the data except SR, BSR and OCR algorithms from the Ricochet family which produce clusters of significantly lower quality. This is mainly due to the inability of these algorithms in finding singleton clusters.

Effectiveness in predicting the correct number of clusters. The results of our experiments, partly shown in Tables 3,4 and 5, 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, MERGE-CENTER, Star, CR and OCR algorithms, can effectively find the right number of datasets with an optimal threshold. The graph-theoretic algorithms Correlation clustering and Markov clustering also find a reasonable number of clusters at lower thresholds.

3.4.3 Running Time

As stated previously, in this work we are focusing mainly on comparing the quality of the 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 (our own and those of others) could be optimized further, and more notably for our study, we have not tried to ensure the time optimization is equitable. Different implementations and machines are used to run these experiments. The table below shows the running times for 8 of the algorithms run using a dataset of 100K records of DBLP titles described in Section 3.1, and with threshold $\theta = 0.4$, giving $386,068$ edges in the similarity graph. PC1 is a Dell 390 Precision desktop with 2.66 GHz Intel Core2 Extreme Quad-Core Processor QX6700, 4GB of RAM running 32-bit Windows Vista. PC2 is an AMD OPTERON 850 2.4GHz, 16 GB of RAM running Red Hat Linux 3.4. PC3 is a Dual Core AMD Opteron Processor 270 (2GHz) with 6GB of memory running Red Hat Linux 2.6. Each experiment is run multiple times to obtain statistical significance.

Algorithm	Time	Lang./Machine
Partitioning	1.790 sec	Java/PC1
CENTER	3.270 sec	Java/PC1
MERGE-CENTER	3.581 sec	Java/PC1
Star	5.9 min	Java/PC1
CCL	83.5 min	Java/PC2
MCL	8.395 sec	C/PC2
MinCut	52.1 min	C & Perl/PC3
ArtPt.	17.563 sec	Perl/PC3

The implementation for the Ricochet algorithms required building a complete similarity graph in memory. Therefore, running the algorithm on the dataset of 100K records required keeping a graph with 100 billion edges in memory which was not possible. We therefore had to limit the size of the dataset for these experiments. We used a dataset of 5K records with threshold $\theta = 0.1$ resulting in 391,706 edges. The running times are shown in the table below.

Algorithm	Time	Lang./Machine
SR	19.687 sec	Java/PC1
BSR	54.115 sec	Java/PC1
CR	9.211 sec	Java/PC1
OCR	8.972 sec	Java/PC1

These results support the scalability of single-pass algorithms as well as MCL and Articulation Point clustering algorithms. Note that we used the original implementation of the MCL algorithm which is highly optimized. Such an optimized implementation was not available for the Correlation Clustering algorithm.

4. SUMMARY AND CONCLUSION

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.

Table 5: Best accuracy values for all the algorithms over medium-error datasets with different distributions

Measure	Group	Part.	CENTER	MergeC	Star	SR	BSR	CR	OCR	CCL	MCL	MinCut	ArtPt.
F1	Uniform	0.910	0.887	0.918	0.892	0.920	0.873	0.910	0.872	0.876	0.921	0.721	0.907
	Zipfian	0.936	0.936	0.938	0.934	0.873	0.463	0.935	0.697	0.929	0.937	0.819	0.934
	Diff.	+0.026	+0.049	+0.020	+0.041	-0.047	-0.411	+0.025	-0.175	+0.054	+0.016	+0.098	+0.027
Cluster#	Uniform	354	472	459	521	504	386	527	454	538	528	665	428
	Zipfian	1018	934	1047.5	933	698.5	158	1061	992	955.25	1021	1038	1067

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. 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.

A summary of the results is presented in Figure 5. Our results using partitioning of the similarity graph (finding the transitive closure of the similarity relation) 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. The Ricochet algorithms produce high quality clusterings when used with uniformly distributed duplicates, but failed in other distributions. All other algorithms were robust to the distribution of the duplicates. 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 Markov clustering as an unconstrained algorithm for duplicate detection and showed that in fact it is among the most accurate algorithms for this task and is also very efficient.

A basic observation here 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 [30], 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.

	Scalability (Current Implementations)	Ability to find the correct number of clusters	Robustness Against		
			Choice of threshold	Amount of Errors	Distribution of errors
Partitioning	High	Low	Low	Low	High
CENTER	High	High	Low	Low	High
MERGE CENTER	High	High	Low	Low	High
Star	Medium	High	Low	Low	High
SR	Low	Medium	High	High	Low
BSR	Low	Low	High	High	Low
CR	Low	High	Medium	High	High
OCR	Low	High	Medium	High	Low
Correlation Clustering	Low	High	Low	Low	High
Markov Clustering	High	High	Medium	Medium	High
Cut Clustering	Low	Low	Low	Low	High
Articulation Point	High	Medium	Low	Low	High

Figure 5: Summary of the results

5. ACKNOWLEDGEMENTS

We thank Derry Tanti Wijaya and Stephane Bressan for kindly making the source code of the Ricochet algorithms available to us. We used the original implementation of the Markov Clustering by Stijn van Dongen. For the Cut Clustering algorithm, we used the push-relabel minimum cut implementation by Andrew Goldberg.

6. REFERENCES

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating Inconsistent Information: Ranking and Clustering. In *ACM Symp. on Theory of Computing (STOC)*, pages 684–693, 2005.

[2] P. Andritsos. *Scalable Clustering of Categorical Data And Applications*. PhD thesis, University of Toronto, Toronto, Canada, September 2004.

[3] A. Arasu, V. Ganti, and R. Kaushik. Efficient Exact Set-Similarity Joins. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 918–929, 2006.

[4] 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.

[5] N. Bansal, A. Blum, and S. Chawla. Correlation Clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[6] N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa. Seeking Stable Clusters In The Blogosphere. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 806–817, Vienna, Austria, 2007.

[7] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling Up All Pairs Similarity Search. In *Int'l World Wide Web Conference (WWW)*, pages 131–140, Banff, Canada, 2007.

[8] I. Bhattacharya and L. Getoor. A Latent Dirichlet Model for Unsupervised Entity Resolution. In *Proc. of the SIAM International Conference on Data Mining (SDM)*, pages 47–58, Bethesda, MD, USA, 2006.

[9] I. Bhattacharya and L. Getoor. Collective Entity Resolution in Relational Data. *IEEE Data Engineering Bulletin*, 29(2):4–12, 2006.

[10] U. Brandes, M. Gaertler, and D. Wagner. Experiments on Graph Clustering Algorithms. In *The 11th Europ. Symp. Algorithms*, pages 568–579. Springer-Verlag, 2003.

[11] S. Brohee and J. van Helden. Evaluation of Clustering Algorithms for Protein-Protein Interaction Networks. *BMC Bioinformatics*, 7:488+, 2006.

[12] M. Charikar, V. Guruswami, and A. Wirth. Clustering with Qualitative Information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.

[13] S. Chaudhuri, V. Ganti, and R. Motwani. Robust Identification of Fuzzy Duplicates. In *IEEE Proc. of the Int'l*

Conf. on Data Eng., pages 865–876, Washington, DC, USA, 2005.

[14] D. Cheng, R. Kannan, S. Vempala, and G. Wang. On a Recursive Spectral Algorithm for Clustering from Pairwise Similarities. Technical Report MIT-LCS-TR-906, MIT LCS, 2003.

[15] F. Chierichetti, A. Panconesi, P. Raghavan, M. Sozio, A. Tiberi, and E. Upfal. Finding Near Neighbors Through Cluster Pruning. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 103–112, Beijing, China, 2007.

[16] 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.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw Hill and MIT Press, 1990.

[18] W. H. Day and H. Edelsbrunner. Efficient Algorithms for Agglomerative Hierarchical Clustering Methods. *Journal of Classification*, 1(1):7–24, 1984.

[19] E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation Clustering In General Weighted Graphs. *Theor. Comput. Sci.*, 361(2):172–187, 2006.

[20] E. A. Dinic. Algorithm for Solution of a Problem Of Maximum Flow in Networks with Power Estimation. *Soviet Math. Dokl*, 11:1277–1280, 1970.

[21] J. Edmonds and R. M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*, 19(2):248–264, 1972.

[22] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *Int. J. Comput. Vision*, 59(2):167–181, 2004.

[23] 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.

[24] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1(4):385–408, 2004.

[25] L. Ford and D. Fulkerson. Maximal Flow Through a Network. *Canadian J. Math*, 8:399–404, 1956.

[26] D. Gibson, R. Kumar, and A. Tomkins. Discovering Large Dense Subgraphs in Massive Graphs. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 721–732, 2005.

[27] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum-Flow Problem. *Journal of the ACM*, 35(4):921–940, 1988.

[28] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *journal*, 17(2-3):107–145, 2001.

[29] O. Hassanzadeh. Benchmarking Declarative Approximate Selection Predicates. Master's thesis, University of Toronto, February 2007.

[30] O. Hassanzadeh and R. J. Miller. Creating Probabilistic Databases from Duplicated Data. Technical Report CSRG-568, University of Toronto, To appear in The VLDB Journal, Accepted on 26 June 2009.

[31] O. Hassanzadeh, M. Sadoghi, and R. J. Miller. Accuracy of Approximate String Joins Using Grams. In *Proc. of the International Workshop on Quality in Databases (QDB)*, pages 11–18, Vienna, Austria, 2007.

[32] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable Techniques for Clustering the Web. In *Proc. of the Int'l Workshop on the Web and Databases (WebDB)*, pages 129–134, Dallas, Texas, USA, 2000.

[33] M. A. Hernández and S. J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[34] L. J. Heyer, S. Kruglyak, and S. Yooseph. Exploring Expression Data: Identification and Analysis of Coexpressed Genes. *Genome Res.*, 9(11):1106–1115, 1999.

[35] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[36] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[37] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, 2004.

[38] Y. J. Kim and J. M. Patel. A Framework for Protein Structure Classification and Identification of Novel Protein Structures. *BMC Bioinformatics*, 7:456+, 2006.

[39] A. D. King. Graph Clustering with Restricted Neighbourhood Search. Master's thesis, University of Toronto, 2004.

[40] J. Kogan. *Introduction to Clustering Large and High-Dimensional Data*. Cambridge Univ. Press, 2007.

[41] C. Li, B. Wang, and X. Yang. VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 303–314, Vienna, Austria, 2007.

[42] 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.

[43] S. Sarawagi and A. Kirpal. Efficient Set Joins On Similarity Predicates. In *ACM SIGMOD Int'l Conf. on the Mgmt. of Data*, pages 743–754, Paris, France, 2004.

[44] N. Slonim. *The Information Bottleneck: Theory And Applications*. PhD thesis, The Hebrew University, 2003.

[45] C. Swamy. Correlation Clustering: Maximizing Agreements Via Semidefinite Programming. In *Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 526–527, New Orleans, Louisiana, USA, 2004.

[46] C. Umans. Hardness of Approximating $\text{Sigma}_2^{\text{p}}$ Minimization Problems. In *Symp. on Foundations of Computer Science (FOCS)*, pages 465–474, 1999.

[47] S. van Dongen. *Graph Clustering By Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[48] J. A. Whitney. Graph Clustering With Overlap. Master's thesis, University of Toronto, 2006.

[49] D. T. Wijaya and S. Bressan. Ricochet: A Family of Unconstrained Algorithms for Graph Clustering. In *Proc. of the Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 153–167, Brisbane, Australia, 2009.

[50] R. Xu and I. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[51] J. Zupan. *Clustering of Large Data Sets*. Research Studies Press, 1982.