Clustering Aggregation

ARISTIDES GIONIS, HEIKKI MANNILA, and PANAYIOTIS TSAPARAS
HIIT Basic Research Unit, Department of Computer Science, University of Helsinki

We consider the following problem: given a set of clusterings, find a clustering that agrees as much as possible with them. This problem, *clustering aggregation*, appears naturally in various contexts. For example, clustering categorical data is an instance of the problem: each categorical variable can be viewed as a clustering of the input rows. Moreover, clustering aggregation can be used as a meta-clustering method to improve the robustness of clusterings. The problem formulation does not require a-priori information about the number of clusters, and it gives a natural way for handling missing values. We give a formal statement of the clustering aggregation problem, discuss related work, and suggest a number of algorithms. For several of the methods we provide theoretical guarantees on the quality of the solutions. We also show how sampling can be used to scale the algorithms for large data sets. We give an extensive empirical evaluation demonstrating the usefulness of the problem and of the solutions.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms: Algorithms

Additional Key Words and Phrases: Data clustering, Clustering categorical data, Clustering aggregation, Correlation clustering

1. INTRODUCTION

Clustering is an important step in the process of data analysis with applications to numerous fields. Informally, clustering is defined as the problem of partitioning data objects into groups (clusters), such that objects in the same group are similar, while objects in different groups are dissimilar. This definition assumes that there is some well defined *quality measure* that captures intra-cluster similarity, and/or inter-cluster dissimilarity. Clustering then becomes the problem of grouping together data objects so that the quality measure is optimized. There is of course an extensive body of literature on clustering methods, see e.g., [Jain and Dubes 1988; Hand et al. 2001; Han and Kamber 2001].

In this paper we consider an approach to clustering that is based on the concept of *aggregation*. We assume that given a set of data objects we can obtain some information on how these objects should be clustered. This information comes in the form of m clusterings $\mathcal{C}_1, \dots, \mathcal{C}_m$. The objective is to produce a single clustering $\mathcal{C}$ that agrees as much as possible with the m clusterings. We define a disagreement

A shorter version of this paper appeared in International Conference on Data Engineering (ICDE), 2005.

	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}$
v_1	1	1	1	1
v_2	1	2	2	2
v_3	2	1	1	1
v_4	2	2	2	2
v_5	3	3	3	3
v_6	3	4	3	3

Fig. 1. An example of clustering aggregation. $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$ are the input clusterings, and $v_1, \ldots, v_6$ are the objects to be clustered. A value k in the entry $(v_i, \mathcal{C}_j)$ means that object v_i belongs to cluster k of the clustering C_j. Column $\mathcal{C}$ is the clustering that minimizes the disagreements with clusterings $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$.

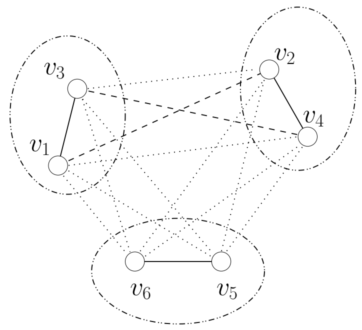

Fig. 2. Correlation clustering instance for the dataset in Figure 1. Solid edges indicate distances of 1/3, dashed edges indicate distances of 2/3, and dotted edges indicate distances of 1. The circles depict the clusters of clustering $\mathcal{C}$ that minimizes the number of disagreements.

between two clusterings $\mathcal{C}$ and $\mathcal{C}'$ as a pair of objects (v, u) such that $\mathcal{C}$ places them in the same cluster, while $\mathcal{C}'$ places them in different clusters, or vice versa. If $d(\mathcal{C}, \mathcal{C}')$ denotes the number of disagreements between $\mathcal{C}$ and $\mathcal{C}'$, then the task is to find a clustering $\mathcal{C}$ that minimizes $\sum_{i=1}^{m} d(\mathcal{C}_i, \mathcal{C})$.

As an example, consider the dataset $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ that consists of six objects, and let $\mathcal{C}_1 = \{\{v_1, v_2\}, \{v_3, v_4\}, \{v_5, v_6\}\}$, $\mathcal{C}_2 = \{\{v_1, v_3\}, \{v_2, v_4\}, \{v_5\}, \{v_6\}\}$, and $\mathcal{C}_3 = \{\{v_1, v_3\}, \{v_2, v_4\}, \{v_5, v_6\}\}$ be three clusterings of V. Figure 1 shows the three clusterings, where each column corresponds to a clustering, and a value i denotes that the tuple in that row belongs in the i-th cluster of the clustering in that column. The rightmost column is the clustering $\mathcal{C} = \{\{v_1, v_3\}, \{v_2, v_4\}, \{v_5, v_6\}\}$ that minimizes the total number of disagreements with the clusterings $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$. In this example the total number of disagreements is 5: one with the clustering $\mathcal{C}_2$ for the pair (v_5, v_6), and four with the clustering $\mathcal{C}_1$ for the pairs $(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4)$. It is not hard to see that this is the minimum number of disagreements possible for any partition of the dataset V.

We define *clustering aggregation* as the optimization problem where, given a set of m clusterings, we want to find the clustering that minimizes the total number of disagreements with the m clusterings. Clustering aggregation provides a general framework for dealing with a variety of problems related to clustering: (*i*) it gives a natural clustering algorithm for categorical data, which allows for a simple treatment of missing values, (*ii*) it handles heterogeneous data, where tuples are defined over incomparable attributes, (*iii*) it determines the appropriate number of clusters and it detects outliers, (*iv*) it provides a method for improving the clustering robustness, by combining the results of many clustering algorithms, (*v*) it allows for clustering of data that is vertically partitioned in order to preserve privacy. We elaborate on the properties and the applications of clustering aggregation in Section 2.

The algorithms we propose for the problem of clustering aggregation take advantage of a related formulation, which is known as *correlation clustering* [Bansal et al. 2002]. We map clustering aggregation to correlation clustering by considering the tuples of the dataset as vertices of a graph, and summarizing the information provided by the m input clusterings by weights on the edges of the graph. The weight of the edge (u, v) is the fraction of clusterings that place u and v in different clusters. For example, the correlation clustering instance for the dataset in Figure 1 is shown in Figure 2. Note that if the weight of the edge (u, v) is less than $1/2$ then the majority of the clusterings place u and v together, while if the weight is greater than $1/2$, the majority places u and v in different clusters. Ideally, we would like to *cut* all edges with weight more than $1/2$, and *not cut* all edges with weight less than $1/2$. The goal in correlation clustering is to find a partition of the vertices of the graph that it cuts as few as possible of the edges with low weight (less than $1/2$), and as many as possible of the edges with high weight (more than $1/2$). In Figure 2, clustering $\mathcal{C} = \{\{v_1, v_3\}, \{v_2, v_4\}, \{v_5, v_6\}\}$ is the optimal clustering.

Clustering aggregation has been previously considered under a variety of names (consensus clustering, clustering ensemble, clustering combination) in a variety of different areas: machine learning [Strehl and Ghosh 2002; Fern and Brodley 2003], pattern recognition [Fred and Jain 2002], bioinformatics [Filkov and Skiena 2003], and data mining [Topchy et al. 2004; Boulis and Ostendorf 2004]. The problem of correlation clustering is interesting in its own right, and it has recently attracted a lot of attention in the theoretical computer science community [Bansal et al. 2002; Charikar et al. 2003; Demaine and Immorlica 2003; Emanuel and Fiat 2003]. We review some of the related literature on both clustering aggregation, and correlation clustering in Section 7.

Our contributions can be summarized as follows.

—We formally define the problem of clustering aggregation, and we demonstrate the connection between clustering aggregation and correlation clustering.

—We present a number of algorithms for clustering aggregation and correlation clustering. We also propose a sampling mechanism that allows our algorithms to handle large datasets. The problems we consider are NP-hard, yet we are still able to provide approximation guarantees for many of the algorithms we propose. For the formulation of correlation clustering we consider we give a combinatorial 3-approximation algorithm.

—We present an extensive experimental study, where we demonstrate the benefits of our approach. Furthermore, we show that our sampling technique reduces the running time of the algorithms, without sacrificing the quality of the clustering.

The rest of this paper is structured as follows. In Section 2 we discuss the various applications of the clustering-aggregation framework, which is formally defined in Section 3. In Section 4 we describe in detail the proposed algorithms for clustering aggregation and correlation clustering, and the sampling-based algorithm that allows us to handle large datasets. Our experiments on synthetic and real datasets are presented in Section 6. Finally, Section 7 contains a review of the related work, and Section 8 is a short conclusion.

2. APPLICATIONS OF CLUSTERING AGGREGATION

Clustering aggregation can be applied in various settings. We will now present some of the main applications and features of our framework.

Clustering categorical data: An important application of clustering aggregation is that it provides a very natural method for clustering categorical data. Consider a dataset with tuples $t_1, \ldots, t_n$ over a set of categorical attributes $A_1, \ldots, A_m$. The idea is to view each attribute A_j as a way of producing a simple clustering of the data: if A_j contains k_j distinct values, then A_j partitions the data in k_j clusters—one cluster for each value. Then, clustering aggregation considers all those m clusterings produced by the m attributes and tries to find a clustering that agrees as much as possible with all of them.

For example, consider a `Movie` database. Each tuple in the database corresponds to a movie that is defined over a set of attributes such as `Director`, `Actor`, `Actress`, `Genre`, `Year`, etc, some of which take categorical values. Note that each of the categorical attributes defines naturally a clustering. For example, the `Movie.Genre` attribute groups the movies according to their genre, while the `Movie.Director` according to who has directed the movie. The objective is to combine all these clusterings into a single clustering.

Methods for aggregating clusterings can also be extended to incorporate domain knowledge, when available. For example, if some attributes are more important than others, then we can increase their influence on the aggregate solution by including multiple copies of the specific attributes in the clustering aggregation. Similarly, if we have some prior knowledge for the relation of the values of a specific attribute (e.g., a hierarchy that renders `thriller` closer to `horror` than to `comedy`) we can incorporate it by adding clusterings that place similar values together. In the movie database example, if the `Movie.Genre` takes the values `thriller, horror`, and `comedy`, we can add to the clustering aggregation a clustering that places all movies with genre `thriller` and `horror` together; this will bias the aggregation algorithm towards merging these two values. In this way we can incorporate the available domain knowledge in the final result in a very natural way.

Clustering heterogeneous data: The clustering aggregation method can be particularly effective in cases where the data are defined over heterogeneous attributes that contain incomparable values. Consider for example the case that there are many numerical attributes whose units are incomparable (say, `Movie.Budget` and `Movie.Year`) and so it does not make sense to compare numerical vectors directly using an L_p-type distance measure. A similar situation arises in the case where the data contains a mix of categorical and numerical values. In such cases the data can be partitioned vertically into sets of homogeneous attributes, obtain a clustering for each of these sets by applying the appropriate clustering algorithm, and then aggregate the individual clusterings into a single clustering.

Missing values: One of the major problems in clustering, and data analysis in general, is the issue of missing values. These are entries that for some reason (mistakes, omissions, lack of information) are incomplete. The clustering aggregation framework provides several ways for dealing with missing values in categorical data. One approach is to average them out: an attribute that contains a missing value in some tuple does not have any information about how this tuple should be clustered, so we should let the remaining attributes decide. When computing the fraction of clusterings that disagree over a pair of tuples, we only consider the attributes that actually have a value on these tuples.

Another approach, which we adopt in this paper, is to assume that given a pair of tuples for which an attribute contains at least one missing value, the attribute tosses a random coin and with probability p, for some chosen $p \in [0, 1]$ it reports the tuples as being clustered together, while with probability $1 - p$ it reports them as being in separate clusters. Each pair of tuples is treated independently. We are then interested in minimizing the *expected* number of disagreements between the clusterings.

Identifying the correct number of clusters: One of the most important features of the formulation of clustering aggregation is that there is no need to specify the number of clusters in the result. The automatic identification of the appropriate number of clusters is a deep research problem that has attracted significant attention (see, e.g., [Schwarz 1978; Hamerly and Elkan 2003; Smyth 2000]). For most clustering approaches the quality (likelihood, sum of distances to cluster centers, etc.) of the solution improves as the number of clusters is increased. Thus, the trivial solution of all singleton clusters is the optimal. There are two ways of handling the problem. The first is to have a hard constraint on the number of clusters, or on their quality. For example, in agglomerative algorithms one can either fix in advance the number of clusters in the final clustering, or impose a bound on the distance beyond which no pair of clusters will be merged. The second approach is to use model selection methods, e.g., Bayesian information criterion (BIC) [Schwarz 1978], or cross-validated likelihood [Smyth 2000] to compare models with different numbers of clusters.

The formulation of clustering aggregation gives one way of automatically selecting the number of clusters. If many input clusterings place two objects in the same cluster, then it will not be beneficial for a clustering-aggregation solution to split these two objects. Thus, the solution of all singleton clusters is not a trivial solution for our objective function. Furthermore, if there are k subsets of data objects in the dataset, such that the majority of the input clusterings places them together, and separates them from the rest, then the clustering aggregation algorithm will correctly identify the k clusters, without any prior knowledge of k. A simple instance is the example in Figures 1 and 2 where the optimal solution $\mathcal{C}$ discovers naturally a set of 3 clusters in the dataset.

Indeed, the structure of the objective function ensures that the clustering aggregation algorithms will naturally settle to the appropriate number of clusters. As we will show in our experimental section, our algorithms take advantage of this feature and for all our datasets they generate clusterings with very reasonable number of clusters. On the other hand, if the user insists on a predefined number of clusters, most of our algorithms can be easily modified to return that specific number of clusters. For example, the agglomerative algorithm described in Section 4 can be made to continue merging clusters until the predefined number is reached.

Detecting outliers: The ability to detect outliers is closely related with the ability to identify the correct number of clusters. If a node is not close to any other nodes, then, from the point of view of the objective function, it would be beneficial to assign that node in a singleton cluster. In the case of categorical data clustering, the scenarios for detecting outliers are very intuitive: if a tuple contains many uncommon values, it is does not participate in clusters with other tuples, and it will be identified as an outlier. Another scenario where it pays off to consider a tuple as an outlier is when the tuple contains common values (and therefore it participates in big clusters in the individual input clusterings) but there is no consensus to a common cluster (for example, a `horror` movie featuring actress `Julia.Roberts` and directed by the "independent" director `Lars.vonTrier`).

Improving clustering robustness: Different clustering algorithms have different qualities and different shortcomings. Some algorithms might perform well in specific datasets but not in others, or they might be very sensitive to parameter settings. For example, the single-linkage algorithm is good in identifying elongated regions but it is sensitive to clusters being connected with narrow strips of points. The k-means method is a widely-used algorithm, but it favors spherical clusters, it is sensitive to clusters of uneven size, and it can get stuck in local optima.

We suggest that by aggregating the results of different clustering algorithms we can improve significantly the robustness and quality of the final clustering. The idea is that different algorithms make different types of mistakes that can be canceled out in the final aggregation. Furthermore, for objects that are outliers or noise, it is most likely that there will be no consensus on how they should be clustered, and thus they will be singled out by the aggregation algorithm. The intuition is similar to performing *rank aggregation* for improving the results of web searches [Dwork et al. 2001]. Our experiments indicate that clustering aggregation can significantly improve the results of individual algorithms.

Privacy-preserving clustering: Consider a situation where a database table is vertically split and different attributes are maintained in different sites. Such a situation might arise in cases where different companies or governmental administrations maintain various sets of data about a common population of individuals. For such cases, our method offers a natural model for clustering the data maintained in all sites as a whole in a privacy-preserving manner, that is, without the need for the different sites to reveal their data to each other, and without the need to rely on a trusted authority. Each site clusters its own data independently and then all resulting clusterings are aggregated. The only information revealed is which tuples are clustered together; no information is revealed about data values of any individual tuples.

3. DESCRIPTION OF THE FRAMEWORK

We begin our discussion of the clustering aggregation framework by introducing our notation. Consider a set of n objects $V = \{v_1, \ldots, v_n\}$. A clustering $\mathcal{C}$ of V is a *partition* of V into k disjoint sets $C_1, \ldots, C_k$, that is, $\bigcup_i^k C_i = V$ and $C_i \cap C_j = \emptyset$ for all $i \neq j$. The k sets $C_1, \ldots, C_k$ are the clusters of $\mathcal{C}$. For each $v \in V$ we use $\mathcal{C}(v)$ to denote the label of the cluster to which the object v belongs, i.e., $\mathcal{C}(v) = j$ if and only if $v \in C_j$. In the sequel we consider m clusterings: we write $\mathcal{C}_i$ to denote the ith clustering, and k_i for the number of clusters of $\mathcal{C}_i$.

In the clustering aggregation problem the task is to find a clustering that minimizes the disagreements with a number of already-existing clusterings. To make the notion more precise, we need to define a measure of disagreement between clusterings. Consider first two objects u and v in V. The following simple 0/1 distance function checks if two clusterings $\mathcal{C}_1$ and $\mathcal{C}_2$ agree on the clustering of u and v.

$$d_{u,v}(\mathcal{C}_1,\mathcal{C}_2) = \begin{cases} 1 & \text{if } \mathcal{C}_1(u) = \mathcal{C}_1(v) \text{ and } \mathcal{C}_2(u) \neq \mathcal{C}_2(v), \\ & \text{or } \mathcal{C}_1(u) \neq \mathcal{C}_1(v) \text{ and } \mathcal{C}_2(u) = \mathcal{C}_2(v), \\ 0 & \text{otherwise.} \end{cases}$$

The distance between two clusterings $\mathcal{C}_1$ and $\mathcal{C}_2$ is defined as the number of pairs of objects on which the two clusterings disagree, that is,

$$d_V(\mathcal{C}_1,\mathcal{C}_2) = \sum_{(u,v)\in V\times V} d_{u,v}(\mathcal{C}_1,\mathcal{C}_2).$$

The clustering aggregation problem can now be formalized as follows.

Problem 1 Clustering aggregation. *Given a set of objects V and m clusterings $\mathcal{C}_1,\ldots,\mathcal{C}_m$ on V, compute a new clustering $\mathcal{C}$ that minimizes the total number of disagreements with all the given clusterings, i.e., it minimizes*

$$D(\mathcal{C}) = \sum_{i=1}^{m} d_V(\mathcal{C}_i,\mathcal{C}).$$

The clustering aggregation problem is also defined in [Filkov and Skiena 2003], where it is shown to be NP-complete using the results of Barthelemy and Leclerc [Barthelemy and Leclerc 1995].

It is easy to show that the distance measure $d_V(\cdot,\cdot)$ satisfies the *triangle inequality* on the space of clusterings.

Observation 1. *Given a set of objects V, and clusterings $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ on V, we have*

$$d_V(\mathcal{C}_1,\mathcal{C}_3) \leq d_V(\mathcal{C}_1,\mathcal{C}_2) + d_V(\mathcal{C}_2,\mathcal{C}_3)$$

Proof. It is sufficient to show that for each pair (u,v) we have $d_{u,v}(\mathcal{C}_1,\mathcal{C}_3) \leq d_{u,v}(\mathcal{C}_1,\mathcal{C}_2) + d_{u,v}(\mathcal{C}_2,\mathcal{C}_3)$. and the lemma follows from the definition of d_V. Since $d_{u,v}$ takes $0/1$ values the only case that the triangle inequality could be violated is when $d_{u,v}(\mathcal{C}_1,\mathcal{C}_3) = 1$ and $d_{u,v}(\mathcal{C}_1,\mathcal{C}_2) = d_{u,v}(\mathcal{C}_2,\mathcal{C}_3) = 0$. However, $d_{u,v}(\mathcal{C}_1,\mathcal{C}_3) = 1$ implies that either $\mathcal{C}_1(u) = \mathcal{C}_1(v)$ and $\mathcal{C}_3(u) \neq \mathcal{C}_3(v)$, or that $\mathcal{C}_1(u) \neq \mathcal{C}_1(v)$ and $\mathcal{C}_3(u) = \mathcal{C}_3(v)$. Assume that $\mathcal{C}_1(u) = \mathcal{C}_1(v)$ and $\mathcal{C}_3(u) \neq \mathcal{C}_3(v)$. Then $d_{u,v}(\mathcal{C}_1,\mathcal{C}_2) = 0$ clusterings $\mathcal{C}_1$ and $\mathcal{C}_2$ must agree on the clustering of u and v; therefore $\mathcal{C}_2(u) = \mathcal{C}_2(v)$. Similarly, $d_{u,v}(\mathcal{C}_1,\mathcal{C}_2) = 0$ implies that $\mathcal{C}_3(u) = \mathcal{C}_3(v)$, which contradicts our assumption that $\mathcal{C}_3(u) \neq \mathcal{C}_3(v)$. The case where $\mathcal{C}_1(u) \neq \mathcal{C}_1(v)$ and $\mathcal{C}_3(u) = \mathcal{C}_3(v)$ is treated symmetrically. □

The algorithms we propose for the problem of clustering aggregation take advantage of a related formulation, which is known as *correlation clustering* [Bansal et al. 2002]. Formally, correlation clustering is defined as follows.

Problem 2 Correlation clustering. *Given a set of objects V, and distances $X_{uv} \in [0,1]$ for all pairs $u,v \in V$, find a partition $\mathcal{C}$ for the objects in*

V that minimizes the score function

$$d(\mathcal{C}) = \sum_{\substack{(u,v) \\ \mathcal{C}(u)=\mathcal{C}(v)}} X_{uv} + \sum_{\substack{(u,v) \\ \mathcal{C}(u)\neq\mathcal{C}(v)}} (1 - X_{uv}).$$

Correlation clustering is a generalization of clustering aggregation. Given the m clusterings $\mathcal{C}_1, \ldots, \mathcal{C}_m$ as input one can construct an instance of the correlation clustering problem by defining the distances X_{uv} appropriately. In particular, let $X_{uv} = \frac{1}{m} \cdot |\{i \mid 1 \leq i \leq m \text{ and } \mathcal{C}_i(u) \neq \mathcal{C}_i(v)\}|$ be the *fraction* of clusterings that assign the pair (u, v) into *different* clusters. For a candidate solution $\mathcal{C}$ of correlation clustering, if $\mathcal{C}$ places u, v in the same cluster it will disagree with mX_{uv} of the original clusterings, while if $\mathcal{C}$ places u, v in different clusters it will disagree with the remaining $m(1 - X_{uv})$ clusterings. Thus, clustering aggregation can be reduced to correlation clustering, and as a result correlation clustering is also an NP-complete problem. We note that an instance of correlation clustering produced by an instance of clustering aggregation is a restricted version of the correlation clustering problem.

It is easy to show that the values X_{uv} obey the triangle inequality.

Observation 2. *For all u, v and w in V, we have that $X_{uw} \leq X_{uv} + X_{vw}$.*

Proof. Define $X^i_{uv} = 1$ if $\mathcal{C}_i(u) \neq \mathcal{C}_i(v)$ and zero otherwise. Then $X_{uv} = \frac{1}{m} \sum_{i=1}^{m} X^i_{uv}$. Therefore, it suffices to show that $X^i_{uw} \leq X^i_{uv} + X^i_{vw}$. The only way that this inequality can be violated is if $X^i_{uw} = 1$ and $X^i_{uv} = X^i_{vw} = 0$. However, the latter equality suggests that u, v, w are all placed in the same cluster, thus reaching a contradiction. □

Since both problems we consider are NP-complete it is natural to seek algorithms with provable approximation guarantees. For the clustering aggregation problem, it is easy to obtain a 2-approximation solution. The idea is to take advantage of the triangle inequality property of the distance measure $d_V(\cdot, \cdot)$. Assume that we are given m objects in a metric space and we want to find a new point that minimizes the sum of distances from the given objects. Then it is a well known fact that selecting the best among the m original objects yields a factor $2(1 - 1/m)$ approximate solution. For our problem, this method suggests taking as the solution to clustering aggregation the clustering $\mathcal{C}_i$ that minimizes $D(\mathcal{C}_i)$. Despite the small approximation factor, this solution is non-intuitive, and we observed that it does not work well in practice.

The above algorithm cannot be used for the problem of correlation clustering—there are no input clusterings to choose from. In general, the correlation clustering problem we consider is not equivalent to the clustering aggregation. There is an extensive literature in the theoretical computer science community on many different variants of the correlation clustering problem; we review some of these results in Section 7. Our problem corresponds to the weighted correlation clustering problem with linear cost functions [Bansal et al. 2002], where the weights on the edges obey the triangle inequality.

4. ALGORITHMS

4.1 Description of the algorithms

In this section we present several algorithms for clustering aggregation. Most of our algorithms approach the problem through the correlation clustering problem, and most of the algorithms are parameter-free.

The BestClustering **algorithm:** This is the simple algorithm that was already mentioned in the previous section. Given m clusterings $\mathcal{C}_1, \ldots, \mathcal{C}_m$, BestClustering finds the input clustering $\mathcal{C}_i$ that minimizes the total number of disagreements $D(\mathcal{C}_i)$. Using the data structures described in [Barthelemy and Leclerc 1995] the best clustering can be found in time $O(mn)$. As discussed, this algorithm yields a solution with approximation ratio of at most $2(1-1/m)$. We can show that this bound is tight, that is, there exists an instance of the clustering aggregation problem, where the algorithm BestClustering produces a solution that is exactly $2(1-1/m)$ worse than the optimal. The proof of the lower bound appears in Section 5.

The algorithm is specific to clustering aggregation – it cannot be used for correlation clustering. An interesting approach to the correlation clustering problem, that might lead to a better approximation algorithm, is to reconstruct m clusterings from the input distance matrix $[X_{uv}]$. Unfortunately, we are not aware of a polynomial-time algorithm for this problem of decomposition into clusterings.

The Balls **algorithm:** The Balls algorithm is inspired by the algorithm in [Charikar et al. 2003] and it works on the correlation clustering problem. It takes as input the matrix of pairwise distances X_{uv}. Equivalently, we view the input as a graph whose vertices are the tuples of a dataset, and the edges are weighted by the distances X_{uv}. The algorithm is defined with an input parameter α, and it is the only algorithm that requires an input parameter. Following the theoretical analysis in Section 5 we can set α to a constant that guarantees a constant approximation ratio. However, different values of α can lead to better solutions in practice.

The intuition of the algorithm is to find a set of vertices that are close to each other and far from other vertices. Given such a set, we consider it to be a cluster, we remove it from the graph, and we proceed with the rest of the vertices. The difficulty lies in finding such a set, since in principle any subset of the vertices can be a candidate. We overcome the difficulty by resorting again to the triangle inequality – this time for the distances X_{uv}. In order to find a good cluster we take all vertices that are close (within a "ball") to a vertex u. The triangle inequality guarantees that if two vertices are close to u, then they are also relatively close to each other. We also note that for the correlation clustering problem it is intuitive that good clusters should be ball-shaped: since our cost function penalizes for long edges that are not cut, we do not expect to have elongated clusters in the optimal solution.

More formally the algorithm is described as follows. It first sorts the vertices in increasing order of the total weight of the edges incident on each vertex. This is a heuristic that we observed to work well in practice. The ordering does not affect the approximation guarantee of the algorithm. At every step, the algorithm picks the first unclustered node u in that ordering. It then finds the set of nodes B that are at a distance of at most $1/2$ from the node u, and it computes the average distance $d(u, B)$ of the nodes in B to node u. If $d(u, B) \leq \alpha$ then the nodes in $B \cup \{u\}$ are considered to form a cluster; otherwise, node u forms a singleton cluster.

We can prove that, when setting $\alpha = \frac{1}{4}$, the cost of solution produced by the BALLS algorithm is guaranteed to be at most 3 times the cost of the optimal clustering. The proof appears in Section 5. In our experiments we have observed that the value $\frac{1}{4}$ tends to be small, as it creates many singleton clusters. For many of our real datasets we have found that $\alpha = \frac{2}{5}$ leads to better solutions. The complexity of the algorithm is $O(mn^2)$ for generating the table and $O(n^2)$ for running the algorithm.

The AGGLOMERATIVE algorithm: The AGGLOMERATIVE algorithm is a standard bottom-up procedure for the correlation clustering problem. It starts by placing every node into a singleton cluster. It then proceeds by considering the pair of clusters with the smallest average distance. The average distance between two clusters is defined as the average weight of the edges between the two clusters. If the average distance of the closest pair of clusters is less than $1/2$ then the two clusters are merged into a single cluster. If there are no two clusters with average distance smaller than $1/2$, then no merging of current clusters can lead to a solution with improved cost $d(\mathcal{C})$. Thus, the algorithm stops, and it outputs the clusters it has created so far.

The AGGLOMERATIVE algorithm has the desirable feature that it creates clusters where the average distance of any pair of nodes is at most $1/2$. The intuition is that the opinion of the majority is respected on average. Using this property we are able to prove that when $m = 3$, the AGGLOMERATIVE algorithm produces a solution with cost at most 2 times that of the optimal solution. The proof appears in Section 5. The complexity of the algorithm is $O(mn^2)$ for creating the matrix plus $O(n^2 \log n)$ for running the algorithm.

The FURTHEST algorithm: The FURTHEST algorithm is a top-down algorithm that works on the clustering correlation problem. It is inspired by the *furthest-first traversal* algorithm, for which Hochbaum and Shmoys [Hochbaum and Shmoys 1985] showed that it achieves a 2-approximation for the clustering formulation of p-centers. As the BALLS algorithm uses a notion of a center to find clusters and repeatedly remove them from the graph, the FURTHEST algorithm uses centers to partition the graph in a top-down fashion.

The algorithm starts by placing all nodes into a single cluster. Then it finds the pair of nodes that are furthest apart, and places them into different clusters. These two nodes become the centers of the clusters. The remaining nodes are assigned to the center that incurs the least cost. This procedure is repeated iteratively: at each step a new center is generated that is the furthest from the existing centers, and the nodes are assigned to the center that incurs the least cost. At the end of each step, the cost of the new solution is computed. If it is lower than that of the previous step then the algorithm continues. Otherwise, the algorithm outputs the solution computed in the previous step. The complexity of the algorithm is $O(mn^2)$ for creating the matrix and $O(k^2 n)$ for running the algorithm, where k is the number of clusters created.

The LocalSearch algorithm: The LocalSearch algorithm is an application of a local-search heuristic to the problem of correlation clustering. The algorithm starts with some clustering of the nodes. This clustering could be a random partition of the data, or it could be obtained by running one of the algorithms we have already described. The algorithm then goes through the nodes and it considers placing them into a different cluster, or creating a new singleton cluster with this node. The node is placed in the cluster that yields the minimum cost. The process is iterated until there is no move that can improve the cost. The LocalSearch can be used as a clustering algorithm, but also as a post-processing step, to improve upon an existing solution.

When considering a node v, the cost $d(v, C_i)$ of assigning a node v to a cluster C_i is computed as follows.

$$d(v, C_i) = \sum_{u \in C_i} X_{vu} + \sum_{u \in S \cap \overline{C_i}} (1 - X_{vu})$$

The first term is the cost of merging v in C_i, while the second term is the cost of *not* merging node v with the nodes not in C_i. We compute $d(v, C_i)$ efficiently as follows. For every cluster C_i we compute and store the cost $M(v, C_i) = \sum_{u \in C_i} X_{vu}$ and the size of the cluster $|C_i|$. Then the distance of v to C_i is

$$d(v, C_i) = M(v, C_i) + \sum_{j \neq i} (|C_j| - M(v, C_j))$$

The cost of assigning node v to a singleton cluster is $\sum_j (|C_j| - M(v, C_j))$.

The running time of the LocalSearch algorithm, given the distance matrix X_{uv}, is $O(Tn^2)$, where Y is the number of local search iterations until the algorithm converges to a solution for which no better move can be found. Our experiments showed that the LocalSearch algorithm is quite effective, and it improves the solutions found by the previous algorithms significantly. Unfortunately, the number of iterations tends to be large, and thus the algorithm is not scalable to large datasets.

4.2 Handling large datasets

The algorithms we described in Section 4.1 take as input the distance matrix, so their complexity is quadratic in the number of data objects in the dataset. The quadratic complexity is inherent in the correlation clustering problem, since the input to the problem is a complete graph. Given a node, the decision of placing the node to a cluster has to take into account not only the cost of merging the node to the cluster, but also the cost of *not* placing the node to the other clusters. Furthermore, the definition of the cost function does not allow for an easy summarization of the clusters, a technique that is commonly used in many clustering algorithms. However, the quadratic complexity makes the algorithms inapplicable to large datasets. We will now describe the algorithm Sampling, which uses sampling to reduce the running time of the algorithms.

The Sampling algorithm is run on top of the algorithms we described Section 4. The algorithm performs a pre-processing and post-processing step that are linear on the size of the dataset. In the pre-processing step the algorithm samples a set of nodes, S, uniformly at random from the dataset. These nodes are given as input to one of the clustering aggregation algorithms. The output is a set of ℓ clusters $\{C_1, ..., C_\ell\}$ of the nodes in S. In the post-processing step the algorithm goes through the nodes in the dataset not in S. For every node it decides whether or to place it in one of the existing clusters, or to create a singleton cluster. In order to perform this step efficiently, we use the same technique as for the LocalSearch algorithm. We observed experimentally that at the end of the assignment phase there are too many singleton clusters. Therefore, we collect all singleton clusters and we run the clustering aggregation again on this subset of nodes.

The size of the sample S is determined so that if there is a large cluster in the dataset the sample will contain at least one node from the cluster with high probability. Large cluster means a cluster that contains a constant fraction of the nodes in the dataset. Using the Chernoff bounds we can prove that sampling $O(\log n)$ nodes is sufficient to ensure that we will select at least one of the nodes in a large cluster with high probability. Note that although nodes in small clusters may not be selected, these will be assigned in singleton clusters in the post-processing step. When clustering the singletons, they will be clustered together. Since the size of these clusters is small this does not incur a significant overhead on cost of the algorithm.

5. THEORETICAL ANALYSIS

In this section we consider some of the algorithms described in Section 4 and we prove guarantees for the cost of the solution they produce with respect to the cost of the optimal solution. For any algorithm Alg, let Alg(I) denote the cost of the algorithm Alg on input I. Also let Opt(I) denote the cost of the optimal solution on input I. Let $|I|$ denote the length of the input. Define $\mathcal{I}$ to be the set of all possible inputs to Alg. We say that the algorithm Alg has approximation ratio $R(\text{Alg}, |I|)$ if for all $I \in \mathcal{I}$, it holds that

$$\text{Alg}(I) \leq R(\text{Alg}, |I|) \cdot \text{Opt}(I)$$

For simplicity we will usually use $R(\text{Alg})$ to denote the approximation ratio of Alg. We are interested in bounding $R(\text{Alg})$ for the different algorithms.

5.1 The BestClustering algorithm

The BestClustering algorithm is an approximation algorithm for Problem 1, the clustering aggregation problem. The input I is a set of m clusterings of n points. The cost function $D(\mathcal{C})$ is the number of disagreements of the output clustering $\mathcal{C}$ with the input clusterings. We know that $R(\text{BestClustering}) \leq 2 - \frac{1}{m}$. We will now prove that this bound is tight.

Theorem 1. *The* BestClustering *algorithm has approximation ratio* $R(\text{BestClustering}) \geq 2 - \frac{1}{m}$ *for Problem 1.*

Proof. In order to prove the lower bound to the approximation ratio of BestClustering it suffices to construct an instance I of the clustering aggregation problem, such that $\frac{\text{BestClustering}(I)}{\text{Opt}(I)} = 2 - \frac{1}{m}$.

Let V be the set of objects and let n denote the size of the set V. We construct m clusterings $\mathcal{C}_1, \ldots, \mathcal{C}_m$ on V as follows. We partition (arbitrarily) the set V

into m subsets $V_1, V_2, \ldots, V_m$ of equal size. For simplicity assume here that n is divisible by m. The clustering $\mathcal{C}_i$ assigns each element of V_i into singleton clusters, while it groups the elements of each set V_j, $j \neq i$, into a single cluster. Formally, the clustering $\mathcal{C}_i$ assigns distinct labels to all elements of the subset V_i, that is, $\mathcal{C}_i(u) \neq \mathcal{C}_i(v)$, for all $u, v \in V_i$. It assigns the same label to all elements in subset V_j, for all $j \neq i$, that is, $\mathcal{C}_i(u) = \mathcal{C}_i(v)$, for all $u, v \in V_j$. Furthermore, for all $j \neq k$, $\mathcal{C}_i(u) \neq \mathcal{C}_i(v)$, for all $u \in V_j$ and $v \in V_k$.

Due to the symmetry in the definition of subsets $V_1, V_2, \ldots, V_m$, and the clusterings $\mathcal{C}_1, \ldots, \mathcal{C}_m$, selecting any clustering $\mathcal{C}_i$ gives the same number of disagreements $D(\mathcal{C}_i)$. Specifically,

$$D(\mathcal{C}_i) = (m-1)\binom{n/m}{2} + (m-1)\binom{n/m}{2} = 2(m-1)\binom{n/m}{2}$$

The first $(m-1)\binom{n/m}{2}$ term is due to the elements of the set V_i. The clustering $\mathcal{C}_i$ assigns a different label to each element in V_i, while each of the other $m-1$ clusterings assigns the same label to all elements in V_i. There are $\binom{n/m}{2}$ pairs, and each of them contributes a disagreement between cluster $\mathcal{C}_i$ and each of the $m-1$ other clusters.

The second $(m-1)\binom{n/m}{2}$ term appears due to the remaining $m-1$ subsets. For each such subset V_j, the clustering $\mathcal{C}_i$ assigns the same label to all elements in V_j. All other clusterings, except for clustering C_j do exactly the same, thus being in agreement with C_i. Clustering $\mathcal{C}_j$ assigns distinct labels to all the elements of V_j, thus generating one disagreement for each pair of elements.

Let $\mathcal{C}^*$ denote the clustering produced by the optimal algorithm. Clustering $\mathcal{C}^*$ creates a cluster for each subset V_i. The total number of disagreements is $D(\mathcal{C}^*) = m\binom{n/m}{2}$. Therefore $D(\mathcal{C}_i) = 2(1 - \frac{1}{m})D(\mathcal{C}^*)$, and $\frac{\textsc{BestClustering}(I)}{\textsc{Opt}(I)} = 2 - \frac{1}{m}$. □

5.2 The Balls algorithm

The Balls algorithm is an approximation algorithm for Problem 2, the correlation clustering problem. The input I to the problem is a set of n points and the pairwise distances X_{uv}. The cost function is $d(\mathcal{C})$ defined in Section 3. We will prove that the approximation ratio of the algorithm is bounded by a constant.

We first prove the following general lemma.

Lemma 1. *For any algorithm* Alg *and any pair of objects u and v*

(a). if $X_{uv} \leq c$ and Alg *assigns u and v in the same cluster, or*

(b). if $X_{uv} \geq 1 - c$ and Alg *assigns u and v in different clusters, then*

the cost payed by Alg *on edge (u, v) is at most $\frac{c}{1-c}$ times the cost of the optimal algorithm for (u, v).*

Proof. In both case (a) and (b) the algorithm Alg pays at most c. If the optimal takes the same decision as Alg, then it pays the same cost. If the optimal takes the opposite decision, then it pays at least $1 - c$, hence the ratio $\frac{c}{1-c}$. □

As an obvious corollary, if $X_{uv} \leq 1/2$ and an algorithm assigns u and v to the same cluster, or if $X_{uv} \geq 1/2$ and an algorithm assigns u and v to different clusters, then the algorithm cannot do worse than the optimal on (u, v).

We are now ready to prove the following theorem. Our proof follows along the lines of the analysis in [Charikar et al. 2003].

THEOREM 2. *For $\alpha = \frac{1}{4}$, the* BALLS *algorithm has approximation ratio* 3

PROOF. We analyze the algorithm by bounding the cost that the algorithm pays for each edge in terms of the cost that the optimal algorithm pays for the same edge. Consider an iteration of the BALLS algorithm, and let u be the node selected to be the center of the ball. We now consider the following cases.

Singleton clusters: First, we consider the case that $C = \{u\}$ is selected to be a singleton cluster. Recall that B is the set of nodes that are within distance $1/2$ from u. For all edges (u, i) with $i \notin B$, we have $X_{ui} \geq 1/2$. Since the algorithm separates u from i, the cost of the optimal cannot be less on each (u, i). The algorithm also splits all edges (u, i) with $i \in B$, so the cost of the algorithm is

$$\sum_{i \in B}(1 - X_{ui}) = |B| - \sum_{i \in B} X_{ui} \leq (1 - \alpha)|B|,$$

where the fact $\sum_{i \in B} X_{ui} \geq \alpha|B|$ follows from the fact that the algorithm chose $\{u\}$ to be a singleton cluster. On the other hand, the optimal algorithm might choose to place u in the same cluster with some vertices $M \subseteq B$. Thus the cost of the optimal for the edges from u to the set B is

$$\sum_{i \in M} X_{ui} + \sum_{i \in B \setminus M}(1 - X_{ui}) \geq \sum_{i \in B} X_{ui} \geq \alpha|B|,$$

where we used the fact that since $i \in B$ we have that $X_{ui} \leq 1/2$, and thus $1 - X_{ui} \geq X_{ui}$. As a result, the approximation ratio within the singleton clusters is at most $R_1 = \frac{(1-\alpha)|B|}{\alpha|B|} = \frac{1-\alpha}{\alpha}$.

Next, we analyze the case where the BALLS algorithm creates the cluster $C = B \cup \{u\}$.

Edges within clusters: For the edges of type (u, i), with $i \in B$, that the algorithm places in the cluster C, we have $X_{ui} \leq 1/2$, so the optimal cannot improve the cost by splitting those edges.

The other type of edges within the cluster $C = B \cup \{u\}$ are edges (i, j) with $i, j \in B$. We order the vertices $i \in B$ in order of increasing distance X_{ui} from the node u. For a *fixed* j we will bound the cost of the edges (i, j) for $i < j$.

If $X_{uj} \leq \beta$, for a constant $\beta < 1/2$ to be specified later, by the triangle inequality for all $i < j$, we have that $X_{ij} \leq X_{ui} + X_{uj} \leq 2\beta$. Therefore, by Lemma 1 the approximation ratio for those edges is at most $R_2 = \frac{2\beta}{1-2\beta}$.

If $X_{uj} > \beta$, let C_j be the set of vertices i with $i < j$. Notice that since the average distance from u to vertices $i \in B$ is less than α, the average distance from u to vertices in C_j is also less than α, since C_j contains a prefix from the list of vertices, ordered in ascending order of their distance X_{ui} from node u. The cost of the algorithm for the edges in C_j is

$$A_j = \sum_{i \in C_j} X_{ij} \leq \sum_{i \in C_j} X_{uj} + \sum_{i \in C_j} X_{ui} \leq \left(\frac{1}{2} + \alpha\right) |C_j|.$$

On the other hand, assume that the optimal algorithm places some vertices $i \in M_j \subseteq C_j$ in the same cluster with j, and the rest of the vertices $i \in S_j = C_j \setminus M_j$ in different clusters than j; thus $|C_j| = |M_j| + |S_j|$. The cost of the optimal algorithm for the edges in C_j can now be written as

$$\begin{aligned}
OPT_j &= \sum_{i \in M_j} X_{ij} + \sum_{i \in S_j} (1 - X_{ij}) \\
&\geq \sum_{i \in M_j} (X_{uj} - X_{ui}) + \sum_{i \in S_j} (1 - X_{uj} - X_{ui}) \\
&= (|M_j| - |S_j|) X_{uj} + |S_j| - \sum_{i \in C_j} X_{ui} \\
&\geq (|M_j| - |S_j|) X_{uj} + |S_j| - \alpha |C_j| \\
&= (|M_j| - |S_j|) X_{uj} + |S_j| - \alpha(|M_j| + |S_j|)
\end{aligned}$$

We now have two cases.

—If $|M_j| < |S_j|$, we use the fact that $X_{uj} \leq 1/2$ or equivalently $(|M_j| - |S_j|)X_{uj} \geq (|M_j| - |S_j|)/2$, and so the cost of the optimal is $OPT_j \geq (|M_j| - |S_j|)/2 + |S_j| - \alpha(|M_j| + |S_j|) = (\frac{1}{2} - \alpha)(|M_j| + |S_j|) = (\frac{1}{2} - \alpha)|C_j|$. In this case the approximation factor is at most $R_3 = \frac{\frac{1}{2}+\alpha}{\frac{1}{2}-\alpha} = \frac{1+2\alpha}{1-2\alpha}$.

—If $|M_j| \geq |S_j|$, we use the fact that $X_{uj} \geq \beta$, implying that the cost of the optimal is $OPT_j \geq \beta(|M_j| - |S_j|) + |S_j| - \alpha(|M_j| + |S_j|) = (\beta - \alpha)|M_j| + (1 - \beta - \alpha)|S_j|$. Selecting $\beta \geq \alpha$, we have that $OPT_j \geq (1 - 2\alpha)|S_j|$.
We now consider difference $A_j - OPT_j$. We have that

$$\begin{aligned}
A_j - OPT_j &= \sum_{i \in C_j} X_{ij} - \left(\sum_{i \in M_j} X_{ij} + \sum_{i \in S_j} (1 - X_{ij}) \right) \\
&= \sum_{i \in S_j} X_{ij} - \sum_{i \in S_j} (1 - X_{ij}) = 2 \sum_{i \in S_j} X_{ij} - |S_j| \\
&\leq 2 \sum_{i \in S_j} 1 - |S_j| = |S_j|
\end{aligned}$$

where the last nequality follows from the fact that $X_{ij} \leq 1$ for all edges (i, j). We now look at the ratio $\frac{A_j - OPT_j}{OPT_j}$. We have that

$$\frac{A_j - OPT_j}{OPT_j} \leq \frac{|S_j|}{(1 - 2\alpha)|S_j|} = \frac{1}{(1 - 2\alpha)}$$

and therefore

$$\frac{A_j}{OPT_j} \leq \frac{2 - 2\alpha}{1 - 2\alpha}$$

In this case, the approximation factor is at most $R_4 = \frac{2-2\alpha}{1-2\alpha}$.

Note that R_2 is a increasing function of β. Since $\beta \geq \alpha$, it takes its minimum value for $\beta = \alpha$ which is $R_2 = \frac{2\alpha}{1-2\alpha}$. We also have that $R_2 \leq R_3$, and $R_2 \leq R_4$ for all $\alpha \in (0, 1/2)$. Therefore, the approximation ratio for the edges within a cluster is at most $\max\{R_3, R_4\}$.

Edges across clusters: Finally, we have to bound the cost of edges going from inside C to clusters outside C. For edges of the type (u, i) with $i \notin C$, we have that $X_{ui} > 1/2$ and the algorithm split those edges, so the optimal cannot perform better on any one those edges. Therefore, we concentrate on edges of the type (i, j) with $i \in C$ and $j \notin C$. In particular $X_{ui} \leq 1/2$ and $X_{uj} > 1/2$. If $X_{uj} \geq \gamma$, for a constant $\gamma > 1/2$ to be specified later, we have that $X_{ij} \geq X_{uj} - X_{ui} \geq \gamma - 1/2$, so, from Lemma 1 the approximation ratio on those edges will be at most $R_5 = \frac{1-(\gamma-1/2)}{\gamma-1/2} = \frac{3/2-\gamma}{\gamma-1/2}$.

In the remaining case $1/2 < X_{uj} < \gamma$, we proceed by *fixing* j and bounding the cost of *all* edges (i, j) for $i \in C$. For some fixed j assume that the optimal algorithm places some vertices $i \in M_j \subseteq C$ in the same cluster with j, and the rest of the vertices $i \in S_j = C \setminus M_j$ in different clusters than j. Again $|C| = |M_j| + |S_j|$. The cost of the algorithm for all edges (i, j) with $i \in C$ is

$$A_j = \sum_{i \in C}(1 - X_{ij}) \leq \sum_{i \in C}(1 - (X_{uj} - X_{ui})) \leq \sum_{i \in C}(1 - X_{uj}) + \sum_{i \in C} X_{ui} \leq \left(\frac{1}{2} + \alpha\right) |C|.$$

The cost of the optimal is bounded from below exactly as in the previous case, that is, $OPT_j \geq (|M_j| - |S_j|)X_{uj} + |S_j| - \alpha(|M_j| + |S_j|)$. If $|M_j| \geq |S_j|$, we use the fact that $X_{uj} > 1/2$, so the cost of the optimal is $OPT_j \geq (\frac{1}{2} - \alpha)|C|$, and the approximation ratio is again R_3.

If $|M_j| < |S_j|$, we use the fact that $X_{uj} < \gamma$, and therefore $OPT_j \geq \gamma(|M_j| - |S_j|) + |S_j| - \alpha(|M_j| + |S_j|) = (\gamma - \alpha)|M_j| + (1 - \gamma - \alpha)|S_j|$. Selecting $\gamma \leq 1 - \alpha$, we have that $OPT_j \geq (1 - 2\alpha)|M_j|$. We consider again the difference $A_j - OPT_j$. We have that

$$\begin{aligned}
A_j - OPT_j &= \sum_{i \in C}(1 - X_{ij}) - \left(\sum_{i \in M_j} X_{ij} + \sum_{i \in S_j}(1 - X_{ij})\right) \\
&= \sum_{i \in M_j}(1 - X_{ij}) - \sum_{i \in M_j} X_{ij} = \sum_{i \in M_j}(1 - 2X_{ij}) \\
&\leq |M_j| - 2\sum_{i \in M_j}(X_{uj} - X_{ui}) \\
&\leq |M_j| - 2\sum_{i \in M_j} X_{uj} + 2\sum_{i \in M_j} X_{ui} \\
&\leq |M_j| - |M_j| + |M_j| = |M_j|
\end{aligned}$$

where the last inequality follows from the fact that $X_{ui} \leq 1/2$ and $X_{uj} > 1/2$.

Similar to before we obtain that

$$\frac{A_j}{OPT_j} \leq \frac{2-2\alpha}{1-2\alpha}$$

Therefore the approximation ration in this case is again at most R_4.

We note that the ratio R_5 is a decreasing function of γ. Since we select $\gamma \leq 1-\alpha$, R_4 takes its minimum value for $\gamma = 1-\alpha$, which is $R_5 = \frac{1/2+\alpha}{1/2-\alpha} = R_3$.

Bringing it all together: The overall approximation ratio of the BALLS algorithm is $R(\text{BALLS}) \leq \max\{R_1, R_3, R_4\}$. The ratios R_1, R_3 and R_4 are functions of the parameter α. We have that $0 \leq \alpha \leq \frac{1}{2}$, and that R_1 is a decreasing function of α, while R_3 and R_4 are increasing functions of α. For $\alpha = \frac{1}{4}$, the values of all three ratios agree to the value 3. Therefore, we conclude that the approximation ratio of the BALLS algorithm is at most 3. □

There are special cases where the BALLS algorithm can perform better. Consider an instance of the correlation clustering problem that is derived when we consider the aggregation of three clusterings. In this case the weights X_{uv} take values in the set $\{0, 1/3, 2/3, 1\}$. Then, for $1/3 \leq \alpha \leq 1/2$ the BALLS algorithm achieves approximation ratio 2. This is due to the fact that in a ball B of radius 1/2 centered at some node u, there are only nodes that are at distance 0 or 1/3 from node u. Selecting α such that $1/3 \leq \alpha \leq 1/2$, we force the algorithm to always create a cluster with the nodes in B. Therefore, by definition, the algorithm takes all edges with weight 0, and breaks all edges with weight 1. For the remaining edges, Lemma 1 guarantees that the approximation ratio is at most 2.

However, this is not so interesting, since even the simple algorithm that merges only the edges of weight zero achieves the same approximation ratio. In general, assume that the X_{uv} satisfy the following property: there exists a value $1/2 \leq c \leq 1$ such that if $X_{uv} \notin \{0,1\}$, then $X_{uv} \in (1-c, c)$. We say that the values X_{uv} are *symmetrically bounded* by c. In this case, Lemma 1 guarantees that the approximation ratio of any algorithm that merges all the zero weight edges, and none of the weight one is at at most $c/(1-c)$. For $c < 3/4$ we guarantee that the approximation ratio is strictly less than 3. Note that for correlation clustering problems derived from clustering aggregation instances the distance values are always symmetrically bounded by $(m-1)/m$, yielding an approximation ratio $m-1$.

5.3 The AGGLOMERATIVE algorithm

For the AGGLOMERATIVE algorithm we can prove that for all input instances such that the X_{uv} values are symmetrically bounded by c, the algorithm achieves approximation ratio $c/(1-c)$. Although, as we noted before, this is achieved by any algorithm that merges all edges of weight zero, and splits all edges of weight one, it is not clear that the AGGLOMERATIVE algorithm satisfies this requirement, since we cannot guarantee that the algorithm will not merge any edge of weight one. We can prove the following theorem.

THEOREM 3. *Assume that for all input instances $I \in \mathcal{I}$ for the correlation clustering problem the values X_{uv} are symmetrically bounded by c, for $1/2 \leq c < 1$. Then the* AGGLOMERATIVE *algorithm has approximation ratio $R(\text{AGGLOMERATIVE}) \leq \frac{c}{c-1}$.*

PROOF. We are going to bound the approximation ratio of AGGLOMERATIVE by bounding the approximation ratio achieved at each step of the algorithm. First we note that in the first steps the AGGLOMERATIVE algorithm will merge all edges of weight zero. This is due to the fact that edges with zero weight appear always in cliques (once again due to triangle inequality). Obviously no approximation error is induced for these merges, thus we can examine the behavior of the algorithm after these merges have been completed. We can thus assume that all subsequent merges involve only edges with weight greater than zero.

Consider now one step of the algorithm after all zero-weight edges have been merged, and let k be the number of edges merged by the algorithm at that step. Let $\{c_1, c_2, ..., c_p\}$ denote the set of all distinct weights that these edges take ($p \leq k$), in decreasing order. Assume that $c_1 = 1$, otherwise Lemma 1 provides the upper bound to the approximation ratio.[1] We have that $c_p > 0$, since we have assumed that all edges of weight zero have already been merged. Let k_i denote the number of edges of weight c_i. The cost paid by the agglomerative algorithm is $A = k_1 + k_2c_2 + \cdots + k_pc_p$. From the definition of the algorithm we have that

$$\frac{k_1 + k_2c_2 + \cdots + k_pc_p}{k_1 + \cdots + k_p} \leq \frac{1}{2}.$$

Solving for k_1 we obtain

$$k_1 \leq (1 - 2c_2)k_2 + (1 - 2c_3)k_3 + \cdots + (1 - 2c_p)k_p$$

Therefore

$$A \leq (1 - c_2)k_2 + (1 - c_3)k_3 + \cdots + (1 - c_p)k_p \leq c(k_2 + k_3 + \cdots + k_p),$$

since for all weights $1 - c \leq c_i \leq c$, and thus $1 - c \leq 1 - c_i \leq c$.

Now let c_q denote the smallest of the weights such that $c_q > 1/2$. The cost of the optimal solution for these edges is at least

$$O = (1 - c_2)k_2 + \cdots (1 - c_q)k_q + c_{q+1}k_{q+1} + \cdots c_pk_p \geq (1 - c)(k_2 + k_3 + \cdots + k_p)$$

Therefore, the approximation ratio is $A/O \leq c/(1 - c)$ which concludes the proof. □

For correlation clustering problems that arise from clustering aggregation problem instances, Theorem 3 guarantees that when merging m clusterings the AGGLOMERATIVE algorithm has approximation ratio at most $m - 1$.

6. EXPERIMENTAL EVALUATION

We have conducted extensive experiments to test the quality of the clusterings produced by our algorithms on a varied collection of synthetic and real datasets. Furthermore, for our SAMPLING algorithm, we have experimented with the quality vs. running-time trade off.

[1]There are cases where a better approximation ratio may be proven, when $c_p < 1$. For example, when X_{uv} takes values from the set $\{1/3, 2/3\}$.

6.1 Improving clustering robustness

The goal in this set of experiments is to show how clustering aggregation can be used to improve the quality and robustness of widely used vanilla clustering algorithms. For the two experiments we are describing we used synthetic datasets of two-dimensional points.

The first dataset is shown in Figure 3. An intuitively good clustering for this dataset consists of the seven perceptually distinct groups of points. We ran five different clustering algorithms implemented in MATLAB: single linkage, complete linkage, average linkage, Ward's clustering, and k-means. For all of the clusterings we set the number of clusters to be 7, and for the rest parameters, if any, we used MATLAB's defaults. The results for the five clusterings are shown in the first five panels of Figure 3. One sees that all clusterings are imperfect. In fact, the dataset contains features that are known to create difficulties for the selected algorithms, e.g., narrow "bridges" between clusters, uneven-sized clusters, etc. The last panel in Figure 3 shows the results of aggregating the five previous clusterings. The aggregated clustering is better than any of the input clusterings (although average-linkage comes very close), and it shows our intuition of how mistakes in the input clusterings can be "canceled out".

In our second experiment the goal is to show how clustering aggregation can be used to identify the "correct" clusters, as well as outliers. Three datasets were created as follows: k^* cluster centers were selected uniformly at random in the unit square, and 100 points were generated from the normal distribution around each cluster center. For the three datasets we used $k^* = 3$, 5, and 7, respectively. An additional 20% of the total number of points were generated uniformly from the unit square and they were added in the datasets. For each of the three datasets we ran k-means with $k = 2, 3, \ldots, 10$, and we aggregated the resulting clusterings, that is, in each dataset we performed clustering aggregation on 9 input clusterings. For lack of space, the input clusterings are not shown; however, most are imperfect. Obviously, when k is too small some clusters get merged, and when k is too large some clusters get split. The results of clustering aggregation for the three datasets are shown in Figure 4. We see that the main clusters identified are precisely the correct clusters. Some small additional clusters are also found that contain only points from the background "noise", and they can be clearly characterized as outliers.

6.2 Clustering categorical data

In this section we use the ideas we discussed in Section 2 for performing clustering of categorical datasets. We used three datasets from the UCI Repository of machine learning databases [Blake and Merz 1998]. The first dataset, **Votes**, contains voting information for 435 persons. For each person there are votes on 16 issues (`yes`/`no` vote viewed as categorical values), and a class label classifying a person as `republican` or `democrat`. There are a total of 288 missing values. The second dataset, **Mushrooms**, contains information on physical characteristics of mushrooms. There are 8,124 instances of mushrooms, each described by 22 categorical attributes, such as shape, color, odor, etc. There is a class label describing if a mushroom is `poisonous` or `edible`, and there are 2,480 missing values in total. Finally, the third dataset, **Census**, has been extracted from the census bureau

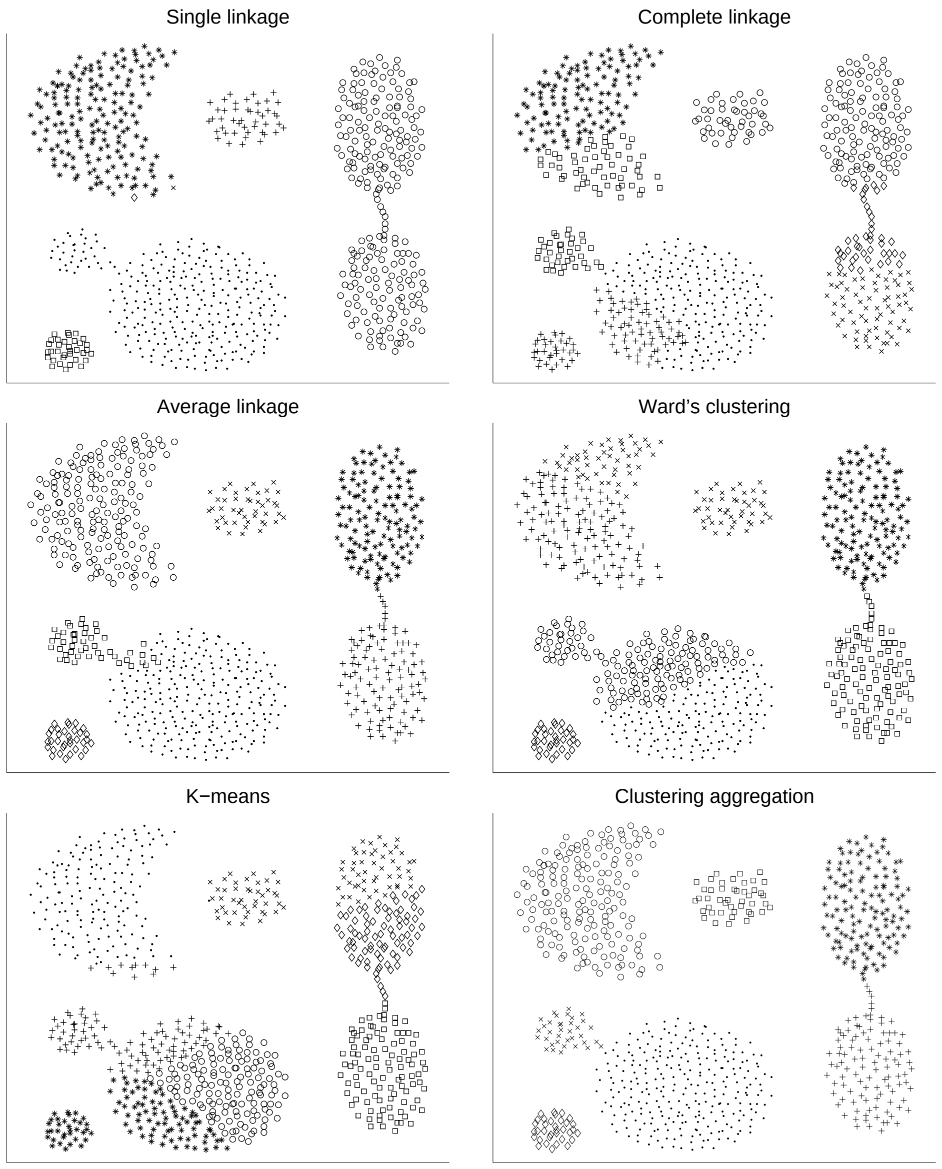

Fig. 3. Clustering aggregation on five different input clusterings. To obtain the last plot, which is the result of aggregating the previous five plots, the Agglomerative algorithm was used.

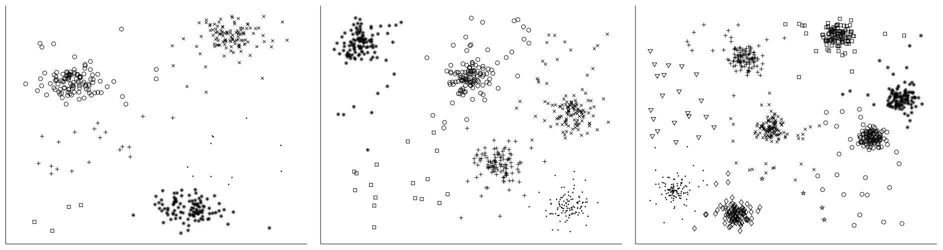

Fig. 4. Finding the correct clusters and outliers.

database, and it contains demographic information on 32,561 people in the US. There are 8 categorical attributes (such as education, occupation, marital status, etc.) and 6 numerical attributes (such as age, capital gain, etc.). Each person is classified according to whether they receive an annual salary of more than $50K or less.

For each of the datasets we perform clustering based on the categorical attributes, and we evaluate the clustering using the class labels of the datasets. The intuition is that clusterings with "pure" clusters, i.e., clusters in which all objects have the same class label, are preferable. Thus, if a clustering contains k clusters with sizes $s_1, \ldots, s_k$, and the sizes of the *majority* class in each cluster are $m_1, \ldots, m_k$, respectively, then we measure the quality of the clustering by an *impurity index* measure, defined as

$$I = \frac{\sum_{i=1}^{k}(s_i - m_i)}{\sum_{i=1}^{k} s_i} = \frac{\sum_{i=1}^{k}(s_i - m_i)}{n}.$$

If a clustering has I value equal to 0 it means that it contains only pure clusters. Notice that clusterings with many clusters tend to have smaller I values—in the extreme case if $k = n$ then $I = 0$ since singleton clusters are pure. We remark that this measure is only indicative of the cluster quality. It is not clear that the best clusters in the dataset correspond to the existing classes. Depending on the application one may be interested in discovering different clusters.

We also run comparative experiments with the categorical clustering algorithm ROCK [Guha et al. 2000], and the much more recent algorithm LIMBO [Andritsos et al. 2004]. ROCK uses the *Jaccard coefficient* to measure tuple similarity, and places a *link* between two tuples whose similarity exceeds a threshold θ. For our experiments, we used values of θ suggested by Guha et al. [Guha et al. 2000] in the original ROCK paper. LIMBO uses information theoretic concepts to define clustering quality. It clusters together tuples so that the conditional entropy of the attribute values within a cluster is low. For the parameter ϕ of LIMBO we used again values suggested in [Andritsos et al. 2004].

The results for the **Votes** and **Mushrooms** datasets are shown in Tables III and IV, respectively. Except for the impurity index (I), we also show the number of clusters of each clustering (k), and the disagreement error (E_D), that is, the objective function in Problem 2 optimized by our algorithms. This is the measure explicitly optimized by our algorithms. Since the clustering aggregation algorithms

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
Poisonous	808	0	1296	1768	0	36	8
Edible	2864	1056	0	96	192	0	0

Table I. Confusion matrix for class labels and clusters found by the AGGLOMERATIVE algorithm on **Mushrooms** dataset.

make their own decisions for the resulting number of clusters, we have run the other two algorithms for the same values of k so that we ensure fairness. Overall the impurity indices are comparable with the exception of LIMBO's impressive performance on **Mushrooms** for $k = 7$ and $k = 9$. Our algorithms achieve the low distance error, with LOCALSEARCH always having the lowest distance error. The distance error for LOCALSEARCH is close to the theoretical lower bound that is computed by considering an idealized algorithm that merges all edges with weight less than $\frac{1}{2}$, and splits all edges with weight more than $\frac{1}{2}$. Furthermore, the attractiveness of the algorithms AGGLOMERATIVE, FURTHEST, and LOCALSEARCH is in the fact that they are completely parameter-free. Neither a threshold nor the number of clusters need to be specified. The number of clusters discovered by our algorithms seem to be very reasonable choices: for the **Votes** dataset, most people vote according to the official position of their political parties, so having two clusters is natural; for the **Mushrooms** dataset, notice that both ROCK and LIMBO achieve much better quality for the suggested values $k = 7$ and $k = 9$, so it is quite likely that the correct number of clusters is around these values. Indicatively, in Table I we present the confusion matrix for the clustering produced by the AGGLOMERATIVE algorithm on the **Mushrooms** dataset.

For the **Census** dataset, clustering aggregation algorithms report about 50-60 clusters. To run clustering aggregation on the **Census** dataset we need to resort to the SAMPLING algorithm. As an indicative result, when the SAMPLING uses the FURTHEST algorithm to cluster a sample of 4,000 persons, we obtain 54 clusters and the impurity index is 24%. ROCK does not scale for a dataset of this size, while LIMBO with parameters $k = 2$ and $\phi = 1.0$ gives impurity index 27.6%. For contrasting these numbers, we mention that supervised classification algorithms (like decision trees and Bayes classifiers) yield classification error between 14 and 21%—but again, we note that clustering is a conceptually different task than classification. We visually inspected some of the smallest of the 54 different clusters, and many corresponded to distinct social groups; for example, male Eskimos occupied with farming-fishing, married Asian-Pacific islander females, unmarried executive-manager females with high-education degrees, etc. An example of such a small cluster is shown in Table II.

6.3 Handling large datasets

In this section we describe our experiments with the SAMPLING algorithm that allows us to apply clustering aggregation to large datasets. First we use the **Mushrooms** dataset to experiment with the behavior of our algorithms as a function of the sample size. As we saw in Table IV, the number of clusters found with the non-sampling algorithms is around 10. When sampling is used, the number of clusters found in the sample remains close to 10. For small sample size, clustering the

WC	Edu	MS	Occ	Rel	Race	Gen	NC
?	1st-4th	MarSpAb	?	Unmar	AIEsk	M	US
Private	10th	NevMar	FarFish	Unmar	AIEsk	M	US
SelfEmp	10th	MarSpAb	AdmCl	Unmar	AIEsk	F	US
SelfEmp	7th-8th	NevMar	FarFish	Unmar	W	M	US
SelfEmp	AsAcdm	Div	CrRep	OwnCh	AIEsk	M	US
SelfEmp	AsAcdm	MarSpAb	FarFish	Unamr	AIEsk	M	US
SelfEmp	AsAcdm	NevMar	FarFish	OthRel	W	M	US
SelfEmp	AsVoc	NevMar	FarFish	NiFam	AIEsk	M	US

Table II. An example of a small cluster representing a distinct social group. The attributes of the table are as follows: WC: Workclass, Edu: Education, MS: Mar-status, Occ: Occupation, Rel: Relationship, Race: Race, Gen: Gender, and NC: Nat-country. The values in the table are as follows: ?: missing value, SelfEmp: Self-emp-not-inc, MarSpAb: Married-spouse-absent, AsAcdm: Assoc-acdm, AsVoc: Assoc-voc, NevMar: Never-married, Div: Divorced, FarFish: Farming-fishing, CrRep: Craft-repair, Unmar: Unmarried, OthRel: Other-relative, OwnCh: Own-child, NiFam: Not-in-family, AIEsk: Amer-Indian-Eskimo, W: White, M: Male, F: Female, AdmCl: Adm-clerical.

Algorithm	k	$I(\%)$	E_D
Class labels	2	0	34,184
Lower bound			28,805
BestClustering	3	15.1	31,211
Agglomerative	2	14.7	30,408
Furthest	2	13.3	30,259
Balls$_{\alpha=0.4}$	2	13.3	30,181
LocalSearch	2	11.9	29,967
ROCK$_{k=2,\theta=0.73}$	2	11	32,486
LIMBO$_{k=2,\phi=0.0}$	2	11	30,147

Table III. Results on **Votes** dataset. k is the number of clusters, I is the impurity index, and E_D is the disagreement error. The lower bound on E_D is computed by considering an algorithm that merges all edges with weight less than $\frac{1}{2}$, and splits all edges with weight greater than $\frac{1}{2}$.

Algorithm	k	$I(\%)$	$E_D(\times 10^6)$
Class labels	2	0	13.537
Lower bound			8.388
BestClustering	5	35.4	8.542
Agglomerative	7	11.1	9.990
Furthest	9	10.4	10.169
Balls$_{\alpha=0.4}$	10	14.2	11.448
LocalSearch	10	10.7	9.929
ROCK$_{k=2,\theta=0.8}$	2	48.2	16.777
ROCK$_{k=7,\theta=0.8}$	7	25.9	10.568
ROCK$_{k=9,\theta=0.8}$	9	9.9	10.312
LIMBO$_{k=2,\phi=0.3}$	2	10.9	13.011
LIMBO$_{k=7,\phi=0.3}$	7	4.2	10.505
LIMBO$_{k=9,\phi=0.3}$	9	4.2	10.360

Table IV. Results on **Mushrooms** dataset.

sample is relatively fast compared to the post-processing phase of assigning the non-sampled points to the best cluster, and the overall running time of the Sampling algorithm is linear. In Figure 5 (left), we plot the running time of the Sampling algorithm, as a fraction of the running time of the non-sampling algorithm, and we show how it changes as we increase the sample size. For a sample of size 1600 we achieve more than 50% reduction of the running time. At the same time, the impurity index of the algorithm converges very fast to the value achieved by the the non-sampling algorithms. This is shown in Figure 5 (middle). For sample size 1600 we have almost the same impurity index, with only half of the running time.

We also measured the running time of the Sampling algorithm for large synthetic datasets. We repeated the configuration of the experiments shown in Figure 4 but on a larger scale. Each dataset consists of points generated from clusters normally distributed around five centers plus an additional 20% of uniformly distributed

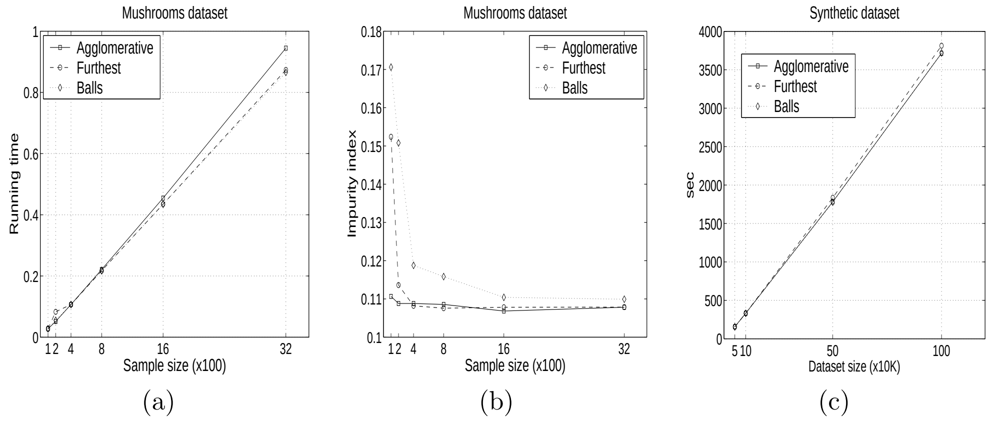

Fig. 5. Scalability Experiments for the Sampling algorithm. **(a)** The running time as a fraction of the time for the whole dataset plotted against the sample size. **(b)** The impurity index as a function of the sample size. **(c)** The running time as a function of the dataset size.

points. We generate datasets of sizes 50K, 100K, 500K, and 1M points. We then cluster the points using MATLAB's k-means for $k = 2, \ldots, 10$, and we run Sampling clustering aggregation on the resulting 9 clusterings. The results are shown in Figure 5 (right). These results are for sample size equal to 1000. Once again, the five correct clusters were identified in the sample, and the running time is dominated by the time to assign the non-sampled points in the clusters of the sample, resulting to the linear behavior shown in the figure.

7. RELATED WORK

A source of motivation for our work is the literature on comparing and merging multiple rankings [Dwork et al. 2001; Fagin et al. 2003]. Dwork et al. [Dwork et al. 2001] demonstrated that combining multiple rankings in a meta-search engine for the Web yields improved results and removes noise (spam). The intuition behind our work is similar. By combining multiple clusterings we improve the clustering quality, and remove noise (outliers).

The problem of clustering aggregation has been previously considered in the machine learning community, under the name *Clustering Ensemble* and *Consensus Clustering*. Strehl and Ghosh [Strehl and Ghosh 2002] consider various formulations for the problem, most of which reduce the problem to a hyper-graph partitioning problem. In one of their formulations they consider the same graph as in the correlation clustering problem. The solution they propose is to compute the best k-partition of the graph, which does not take into account the penalty for merging two nodes that are far apart. All of their formulations assume that the correct number of clusters is given as a parameter to the algorithm.

Fern and Brodley [Fern and Brodley 2003] apply the clustering aggregation idea to a collection of soft clusterings they obtain by random projections. They use an agglomerative algorithm similar to ours, but again they do not penalize for merging dissimilar nodes. Fred and Jain [Fred and Jain 2002] propose to use a single linkage algorithm to combine multiple runs of the k-means algorithm. Dana Cristofor and Dan Simovici [Cristofor and Simovici 2001] observe the connection between clustering aggregation and clustering of categorical data. They propose information theoretic distance measures, and they propose genetic algorithms for finding the best aggregation solution. Boulis and Ostendorf [Boulis and Ostendorf 2004] use Linear Programming to discover a correspondence between the labels of the individual clusterings and those of an "optimal" meta-clustering. Topchy et al. [Topchy et al. 2004] define clustering aggregation as a maximum likelihood estimation problem, and they propose an EM algorithm for finding the consensus clustering. Filkov and Skiena [Filkov and Skiena 2003] consider the same distance measure between clusterings as ours. They propose a simulating annealing algorithm for finding an aggregate solution, and a local search algorithm similar to ours. They consider the application of clustering aggregation to the analysis of microarray data.

There is an extensive literature in the field of theoretical computer science for the problem of correlation clustering. The problem was first defined by Bansal et al. [Bansal et al. 2002]. In their definition, the input is a complete graph with +1 and -1 weights on the edges. The objective is to partition the nodes of the graph so as to minimize the number of positive edges that are cut, and the number of negative edges that are not cut. The best known approximation algorithm for this problem is by Charikar et al. [Charikar et al. 2003] who give an LP-based algorithm that achieves a 4 approximation factor. When the edge weights are arbitrary, the problem is equivalent to the multi-way cut, and thus there is a tight $O(\log n)$-approximation bound [Demaine and Immorlica 2003; Emanuel and Fiat 2003]. If one considers the corresponding maximization problem, that is, maximize the agreements rather than minimize disagreements, then the situation is much better. Even in the case of graphs with arbitrary edge weights there is a 0.76-approximation algorithm using semi-definite programming [Charikar et al. 2003; Swamy 2004].

Recently, Ailon et al. [Ailon et al. 2005] considered a variety of correlation clustering problems. They proposed an algorithm very similar to the Balls algorithm, and they showed that if the weights obey the *probability condition* (that is, for every edge (i, j), the cost for taking that edge is $X_{ij} \in [0, 1]$, while the cost for splitting an edge is $1 - X_{ij}$), then their algorithm achieves expected approximation ratio 5. If the weights X_{ij} obey also the triangular inequality, then the algorithm achieves expected approximation ratio 2. For the clustering aggregation problem they show that choosing the best solution between their algorithm and the BestClustering algorithm yields a solution with expected approximation ratio 11/7.

Our work differs from the work of Ailon et al. [Ailon et al. 2005] in that we provide a worst case approximation ratio for the correlation clustering problem. Furthermore, we investigate experimentally the performance of our algorithms for various applications, such as clustering of categorical data, clustering robustness, and finding the correct number of clusters. For the problem of categorical clustering we compare our algorithms with various existing algorithms to demonstrate the benefits of our approach.

8. CONCLUDING REMARKS

In this paper we considered the problem of clustering aggregation. Simply stated, the idea is to cluster a set of objects by trying to find a clustering that agrees as much as possible with a number of already-existing clusterings. We motivated the problem by describing in detail various applications of clustering aggregation including clustering categorical data, dealing with heterogeneous data, improving clustering robustness, and detecting outliers. We formally defined the problem and we showed its connection with the problem of correlation clustering. We proposed various algorithms for both the clustering aggregation and the clustering correlation problem, including a sampling algorithm that allows us to handle large datasets with no significance loss in the quality of the solutions. We also analyzed the algorithms theoretically, providing approximation guarantees whenever possible. Finally, we verified the intuitive appeal of the proposed approach and we studied the behavior of our algorithms with experiments on real and synthetic datasets.

Acknowledgments We thank Alexander Hinneburg for many useful comments, and Periklis Andritsos for pointers to code, data, and his help with the experiments.

REFERENCES

Ailon, N., Charikar, M., and Newman, A. 2005. Aggregating inconsistent information: ranking and clustering. In *ACM Symposium on Theory of Computing (STOC)*.

Andritsos, P., Tsaparas, P., Miller, R. J., and Sevcik, K. C. 2004. LIMBO: Scalable clustering of categorical data. In *EDBT*.

Bansal, N., Blum, A., and Chawla, S. 2002. Correlation clustering. In *FOCS*.

Barthelemy, J.-P. and Leclerc, B. 1995. The median procedure for partitions. *DIMACS Series in Discrete Mathematics*.

Blake, C. L. and Merz, C. J. 1998. UCI repository of machine learning databases.

Boulis, C. and Ostendorf, M. 2004. Combining multiple clustering systems. In *PKDD*.

Charikar, M., Guruswami, V., and Wirth, A. 2003. Clustering with qualitative information. In *FOCS*.

Cristofor, D. and Simovici, D. A. 2001. An information-theoretical approach to genetic algorithms for clustering. Tech. Rep. TR-01-02, UMass/Boston.

Demaine, E. D. and Immorlica, N. 2003. Correlation clustering with partial information. In *APPROX*.

Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. 2001. Rank aggregation methods for the Web. In *WWW*.

Emanuel, D. and Fiat, A. 2003. Correlation clustering: Minimizing disagreements on arbitrary weighted graphs. In *ESA*.

Fagin, R., Kumar, R., and Sivakumar, D. 2003. Comparing top k lists. In *SODA*.

Fern, X. Z. and Brodley, C. E. 2003. Random projection for high dimensional data clustering: A cluster ensemble approach. In *ICML*.

Filkov, V. and Skiena, S. 2003. Integrating microarray data by concensus clustering. In *International Conference on Tools with Artificial Inteligence*.

Fred, A. and Jain, A. K. 2002. Data clustering using evidence accumulation. In *ICPR*.

Guha, S., Rastogi, R., and Shim, K. 2000. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems 25,* 5, 345–366.

Hamerly, G. and Elkan, C. 2003. Learning the k in k-means. In *NIPS*.

Han, J. and Kamber, M. 2001. *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Hand, D., Mannila, H., and Smyth, P. 2001. *Principles of Data Mining*. The MIT Press, Cambridge, Massachusetts.

Hochbaum, D. and Shmoys, D. 1985. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 180–184.

Jain, A. K. and Dubes, R. C. 1988. *Algorithms for Clustering Data.* Prentice-Hall.

Schwarz, G. 1978. Estimating Dimension of a Model. *Annals of Statistics 6*, 461–464.

Smyth, P. 2000. Model selection for probabilistic clustering using cross-validated likelihood. *Statistics and Computing 10,* 1, 63–72.

Strehl, A. and Ghosh, J. 2002. Cluster ensembles — A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*.

Swamy, C. 2004. Correlation clustering: maximizing agreements via semidefinite programming. In *SODA*.

Topchy, A., Jain, A. K., and Punch, W. 2004. A mixture model of clustering ensembles. In *SDM*.