Minimum Spanning Tree Partitioning Algorithm for Microaggregation

Michael Laszlo and Sumitra Mukherjee, *Member*, *IEEE*

Abstract—This paper presents a clustering algorithm for partitioning a minimum spanning tree with a constraint on minimum group size. The problem is motivated by microaggregation, a disclosure limitation technique in which similar records are aggregated into groups containing a minimum of k records. Heuristic clustering methods are needed since the minimum information loss microaggregation problem is NP-hard. Our MST partitioning algorithm for microaggregation is sufficiently efficient to be practical for large data sets and yields results that are comparable to the best available heuristic methods for microaggregation. For data that contain pronounced clustering effects, our method results in significantly lower information loss. Our algorithm is general enough to accommodate different measures of information loss and can be used for other clustering applications that have a constraint on minimum group size.

Index Terms—Clustering, partitioning, minimum spanning tree, microdata protection, disclosure control.

1 Introduction

Statistical agencies have the responsibility of disseminating data for legitimate statistical purposes while protecting the confidentiality of individual data subjects. Several methods have been proposed for disclosure limitation (see [1] or [12] for a survey of disclosure control methods). Microaggregation is a disclosure limitation technique that is used to protect microdata files containing records on individual data subjects. Under microaggregation, records are aggregated into groups; actual values of individual records are replaced by group means prior to release. By ensuring that each group contains a minimum of k records, the confidentiality of individual data subjects is protected. The objective is to group similar records together so that the replacement of actual values by group means results in minimum information loss.

While an efficient polynomial algorithm exists for optimal univariate microaggregation [6], minimum information loss microaggregation for multivariate data is known to be NP-hard [9]. Hence, several heuristic methods for multivariate microaggregation have been proposed [4]. These methods include using univariate projections of multivariate data, forming fixed-sized groups, and adapting Ward's hierarchical clustering strategy [11]. Results from computational experiments are inconclusive about the dominance of any of these techniques and there is a consensus among researchers that alternative heuristic clustering approaches must be investigated for the minimum information loss microaggregation problem [4]. New heuristic methods for microaggregation may be useful for practitioners if they are at least as efficient or effective as available methods. We present an algorithm that has the same complexity as the multivariate k-Ward microaggregation algorithm proposed in [4] and experiments indicate that information losses resulting from our method are no worse than those reported in that study.

Many data clustering algorithms have been proposed in the literature (see [7] for a comprehensive review). Clustering algorithms may be good candidates for microaggregation if they can be adapted to enforce the group size constraint. Among the most commonly used methods is one that is based on partitioning a minimum spanning tree (MST) [13]. The standard MST partitioning algorithm first constructs the MST of the complete graph $G = (V, E)$, where each node in V corresponds to a data point and the length of each edge is the Euclidean distance between its two endpoints, and then removes the $c - 1$ longest edges to produce c trees, each of which corresponds to a cluster. Intuitively, the longest edges—which are removed—separate the natural clusters, and the shortest edges—which are retained—connect close data points within natural clusters.

The microaggregation problem differs from the classical clustering problem in two respects. In classical clustering, the number of desired clusters is given a priori whereas, in microaggregation, it is not. Second, in classical clustering, there is no constraint on the size of clusters whereas, in microaggregation, each cluster must contain no fewer than k data points where k is given. The standard MST partitioning algorithm, which removes edges from the MST in order of decreasing length until the specified number of clusters results, does not solve the microaggregation problem since it does not accommodate the group size constraint. To adapt this algorithm for microaggregation, we modify the strategy for selecting edges for deletion from the MST: Edges are inspected in the order of decreasing length and removed only if each tree that results from the edge's removal has at least k nodes. The resulting trees then discern pronounced clusters while also satisfying the minimum group size constraint.

• *The authors are with the Graduate School of Computer and Information Sciences, Nova Southeastern University, 3301 College Ave., Fort Lauderdale, FL 33314-7796. E-mail: {mjl, sumitra}@nova.edu.*

Manuscript received 26 Feb. 2004; revised 30 Aug. 2004; accepted 17 Feb. 2005; published online 18 May 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0058-0204.

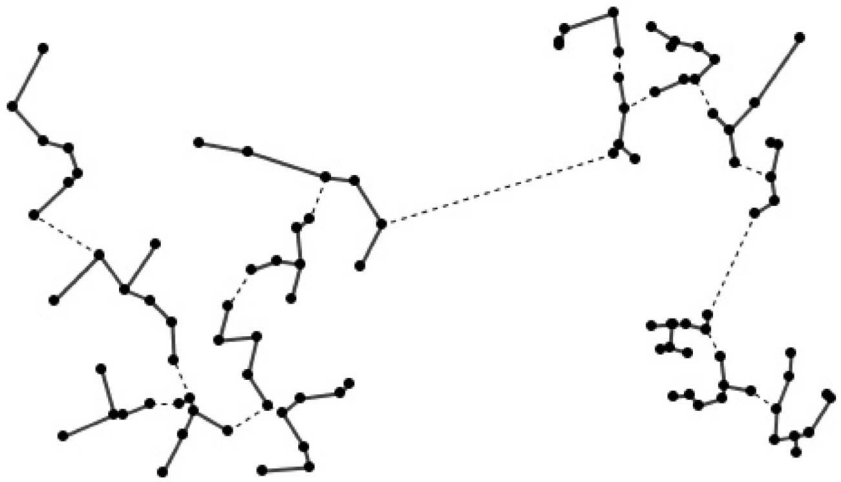


Fig. 1. An MST partition.

Fig. 1 illustrates an MST partition of 102 data points with group size constraint set to 5. The edges that get removed are dashed and those that are retained are solid; a partition into 16 groups results.

Our paper presents an efficient MST-based clustering technique that enforces the minimum group size constraint. Results of computational experiments indicate that our MST partitioning algorithm for microaggregation is sufficiently efficient to be practical for large data sets and yields results that are comparable to the best available heuristic methods. For data sets that contain pronounced clustering effects, our algorithm results in significantly lower information loss.

We formally define the microaggregation problem in Section 2 and present our MST partitioning algorithm in Section 3. Section 4 presents experimental results with data sets used in previous studies and with simulated data that contain well-defined clusters. Section 5 concludes with a discussion on the contributions of this article and identifies possible extensions of this work. Appendix A presents recurrence relations for the size of the search space, and Appendix B characterizes the structure of the trees resulting from our MST partitioning algorithm.

2 The Microaggregation Problem

The microaggregation problem can be formally stated as follows: Consider a microdata set with n records and p numerical attributes. A p-dimensional point represents each record. Microaggregation involves partitioning the n data points such that each group contains at least k points and each point is contained in exactly one group. That is,

$$n_j \geq k, \quad \text{for } j = 1, 2, \ldots, g, \tag{1}$$

$$\sum_{j=1}^{g} n_j = n, \tag{2}$$

where n_j is the number of observations in group j.

Using the Euclidean distance between vectors as a measure of similarity, our objective is to minimize the sum of the within-group squared error:

$$SSE = \sum_{j=1}^{g} \sum_{i=1}^{n_j} (x_{ij} - \bar{x}_j)'(x_{ij} - \bar{x}_j), \tag{3}$$

where x_{ij} is the ith point in the jth group, $i = 1, 2, \ldots, n_j$, $j = 1, 2, \ldots, g$, and $\bar{x}_j = \frac{1}{n_j}\sum_{i=1}^{n_j} x_{ij}$ is the centroid for group j.

The microaggregation problem hence involves partitioning the n points into groups so as to minimize the objective function (3) subject to (1) and (2). Minimizing *SSE* is equivalent to minimizing a standardized information loss measure that is commonly used in the microaggregation literature. This information loss measure is defined as $L = SSE/SST$, where *SST* is the squared error that would result if all records were included in a single group; that is, $SST = \sum_{i=1}^{n}(x_i - \bar{x})'(x_i - \bar{x})$, where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

Domingo-Ferrer and Mateo-Sanz [4] have shown that in an optimal partition, no group will contain more than $2k-1$ records since groups of size $2k$ or larger can always be partitioned further to reduce information loss. However, even accounting only for partitions containing groups of size between k and $2k-1$, the number of partitions is exponential in the number of data points. Appendix A presents recurrence relations that characterize the number of partitions whose groups have a size between k and $2k-1$ inclusive. We note that an exhaustive examination of all possible partitions with typical values of $k(3 \leq k \leq 10)$ is clearly impractical for any application containing even as few as a 100 records. Hence, we present our MST partitioning algorithm as a useful heuristic in lieu of exhaustive search.

3 The Algorithm

A tree is said to be *admissible* if it satisfies the group size constraint, that is, if it contains no fewer than k nodes. A forest is *admissible* if all of its trees are admissible. Given an admissible forest F, an edge $e \in F$ is said to be *removable* if the forest $F - \{e\}$ obtained by removing edge e from F is admissible. When a removable edge is deleted from an admissible tree, each of the two trees that result is admissible.

Our clustering algorithm proceeds in three phases:

1. *MST construction*: Construct the minimum spanning tree F over the data points using Prim's algorithm.

2. *Edge cutting*: Iteratively delete removable edges from F in the order of decreasing length.
3. *Cluster formation*: Traverse the resulting forest F to assign each data point to a cluster.

Assuming that the number $|V|$ of data points is at least k at the outset, the initial forest MST is an admissible one-tree forest, and each iteration performed in step 2 transforms an admissible forest of n trees into an admissible forest of $n+1$ trees. The clusters formed in step 3 necessarily satisfy the group size constraint. We elaborate each of these phases in the sections that follow.

3.1 MST Construction

Our algorithm relies on Prim's method for construction of the MST over the data points V under the Euclidean metric (e.g., see [2] and [10]). Initially, Prim's method selects an arbitrary data point $v \in V$ and builds a minimum-cost tree T spanning $S = \{v\}$; here, T consists of the point $\{v\}$ and no edges. The method proceeds iteratively: In each iteration, it selects some point $u \in V - S$ that is closest to set S; where point $w \in S$ is closest to u, it adds the edge (u, w) to tree T and adds point u to set S. The process continues until $S = V$, at which time T represents the MST spanning point set V as a whole. In each iteration, T is a minimum-cost tree spanning the data points in set S. Prim's method grows a minimum-cost tree from a single data point into a minimum-cost tree that spans the entire set V of data points.

We represent the successive forest topologies, including the initial MST, as a *rooted forest*: Each nonroot node points to its parent node in the tree to which it belongs, and each root node points to the special value *null*. In addition, each node n contains a *descendant count* denoting the number of nodes in the subtree rooted at node n including n itself.

MST construction augments the standard implementation of Prim's method in two ways. First, as each MST edge is discovered, it is inserted into a priority queue based on edge length; this priority queue is used during the subsequent edge-cutting phase to obtain the edges in decreasing length order. Second, the descendant counts for all nodes are computed.

One method for calculating descendant counts is quite simple: Each time the tree T under construction is extended by a data point u during the course of Prim's method, increment by one the descendant count of every node along the path from u to the root of tree T. This approach takes time proportional to the sum of the node depths in the resulting MST, which is $O(|V|^2)$. A more efficient method postpones the calculation of descendant counts until the MST is fully constructed. In this method, when a new node is added to tree T during the course of Prim's method, the node is also pushed onto a stack and its descendant count is initialized to one. When the MST is complete, we process the nodes in the stack: For each node p, increment the descendant count of p's parent by the descendant count of p, thereby accounting for the data points contributed by the subtree rooted at p. Correctness follows from the fact that every node appears higher in the stack than its parent. By the time a node p's descendant count is added to its parent's count, all of p's children will have already contributed their descendant counts to p. This approach computes all descendant counts in $O(|V|)$ time since each of the $|V|$ nodes in the stack requires constant-time processing.

Construction of the MST is dominated by Prim's algorithm since calculation of descendant counts takes $O(|V|)$ time. If Fibonacci heaps are used, the running time for Prim's algorithm is $O(|E| + |V| \lg |V|)$, where $|E|$ denotes the number of edges in the original graph whose MST is sought [2]. This "original graph" is effectively the complete graph over the $|V|$ data points, implying that $|E| \in O(|V|^2)$, so MST construction takes $O(|V|^2)$ time.[1]

3.2 Cutting Edges

The edge-cutting phase iteratively visits every MST edge in length order, from longest to shortest, and deletes the removable edges while retaining the remaining edges. Edges are obtained in length order from the priority queue constructed during the previous phase. What results is a series of forests F, starting with the MST, each of which possesses one more tree than its predecessor.

Descendant counts are used to determine whether an edge is removable, that is, whether the forest that would result from its removal is admissible. Consider the edge $e = (org, dest)$, directed from node org toward its parent node $dest$. The size of the tree to which org belongs in the forest $F - \{e\}$ is given by C_{org}, where C_{org} denotes the descendant count of node org in the forest F. Moreover, the size of the tree to which node $dest$ belongs in the forest $F - \{e\}$ is given by $C_{root} - C_{org}$, where C_{root} denotes the descendant count of the root node of the tree to which both org and $dest$ belong in the forest F. Accordingly, edge e is removable if and only if the following two conditions hold:

1. $C_{org} \geq k$ and
2. $C_{root} - C_{org} \geq k$.

This leads to the following pseudocode routine for deciding whether an edge $(org, dest)$ is removable:

```
boolean isRemovable( DataPoint org, DataPoint
                     dest ) {
  if (Corg < k)
    return false;
  DataPoint root ← getRoot( dest );
  return ((Croot − Corg) ≥ k);
}
```

The procedure `getRoot` takes a node n and returns the root node of n's tree by following parent pointers.

When a removable edge $e = (org, dest)$ is deleted, it is necessary to update the descendant counts along the path of ancestors from node $dest$ to the root node of its tree. Specifically, the descendant count of each node along this path gets decremented by C_{org} (see Fig. 2). Accordingly, the

1. In the special case when the data points have dimension two, it is possible to first construct the Delaunay graph of the data points and then run Prim's algorithm on the Delaunay graph (which is known to contain an MST as a subgraph). Construction of the Delaunay graph takes $O(|V| \lg |V|)$ time; morever, since the Delaunay graph is planar, the number of edges it contains is $O(|V|)$. Taking this approach, construction of the MST improves to $O(|V| \lg |V|)$ in the special case when the data is two-dimensional.

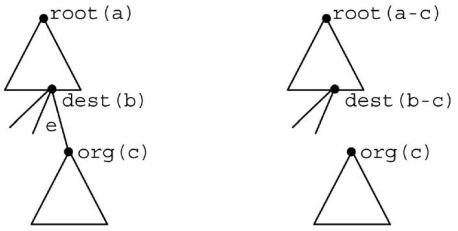


Fig. 2. Removing edge e from the rooted forest, with descendant counts in parentheses.

edge-cutting phase of our algorithm is carried out by the following procedure where the function `getNext` extracts the MST edges from the priority queue `edges` in descending-length order:

```
void cutEdges( PriorityQueue edges ) {
  while (edges ≠ ∅) {
    (org, dest) ← getNext (edges);
    if (isRemovable(org,dest)) {
      setParent(org, null); // org becomes a
                               root node
      u ← dest;          //update counts in
                           dest's tree
      while (u ≠ null) {
        C_u ← C_u − C_org;
        u ← getParent(u);
      }
    }
  }
}
```

The trees produced by the edge-cutting phase are *irreducible* in the sense that they contain no removable edges: An irreducible tree T contains no edge e such that each of the two trees in the graph $T - \{e\}$ has size at least k. We formally characterize the structure of irreducible trees in Appendix B.

The edge-cutting phase runs in $O(|V|^2)$ time in the worst-case sense. This behavior is due to the $O(|V|)$ time it can take to decide whether an edge is removable. This worst-case behavior is realized, for instance, when the MST takes the form of an irreducible chain of data, implying that no edge is removable. On the positive side, average node depth in the forest decreases as edges are removed. Moreover, one can produce an MST that tends toward lower average node depth by seeding Prim's algorithm with a data point that lies near the centroid of point set V.

3.3 Forming Clusters

We form clusters by following parent pointers in the rooted forest. Our method implements a loop that iteratively applies a procedure, which we'll call `classifyPoint`, to every data point p in arbitrary order. The `classifyPoint` procedure assigns p to a cluster and returns p's cluster. If point p has already been classified, its cluster is returned. Else if p is a root node, a new cluster is constructed to which p is assigned. Otherwise, the parent node of p is classified recursively, and then p is assigned to its parent's cluster. In effect, to classify a data point p, we recursively traverse the path of ancestor nodes from p until reaching some node q that has either already been assigned to a cluster or is a root node, and then we assign every node along this path to the cluster to which q belongs. The pseudocode for this procedure follows:

```
Cluster classifyPoint( DataPoint p ) {
  Cluster C ← getCluster( p );
  if (C ≠ null)        // p belongs to a cluster
    return C;
  else if ( getParent(p) = null) // p is a root
                                    node
    C ← create new cluster;
  else
    C ← classifyPoint( getParent(p) );
  C ← C ∪ {p};
  return C;
}
```

We show that cluster formation takes $O(|V|)$ time by bounding the total number of times that `classifyPoint` is called. Since every data point gets assigned to a cluster exactly once, `classifyPoint` is called exactly $|V|$ times on as-yet-unclassified points. How many times is it called on already-classified data points? This occurs at most $|V|$ times due to the top-level loop that invokes `classifyPoint` once on each data point of V. In addition, `classifyPoint` can be called on an already-classified point p, where p is the parent of a point that is in the process of being classified, but this can occur no more than $|V|$ times (the call to `classifyPoint` can be charged to the child node being classified). In total, no more than $3|V| \in O(|V|)$ constant-time calls are made to `classifyPoint`.

4 Experimental Results

In this section, we describe our data sets, explain our experimental methods, and present our results.

4.1 Data Sets

We test our method using three data sets that have been used in previous studies:

1. The *Tarragona* data set contains 834 records with 13 variables [4].
2. The *Census* data set contains 1,080 records with 13 variables [5].
3. The *Creta* data set contains 1,080 records with 13 variables [3].

To further investigate the performance of our algorithm, we also test it with 29 simulated data sets generated using the process described in [8]: c random cluster centers are generated uniformly over the d-dimensional hypercube $[-1, 1]^d$. The number of points n_i in cluster i is randomly selected from a uniform distribution between *min_cluster_size* and *max_cluster_size*. The n_i points in cluster i are generated around the cluster center such that each of the d coordinates independently follows a Gaussian distribution with standard deviation σ.

This process was used to generate the following four classes of data sets:

TABLE 1
Data Sets to Investigate the Effect of Natural Clustering

Data Set	SIM_1	SIM_2	SIM_3	SIM_4	SIM_5	SIM_6	SIM_7	SIM_8
σ	0.010	0.025	0.050	0.075	0.100	0.150	0.200	0.300
n	10943	11239	11343	10862	11038	10513	11185	11059

TABLE 2
Data Sets to Investigate Scalability with Respect to Data Set Size

Data Set	SIM_9	SIM_10	SIM_11	SIM_12	SIM_13	SIM_14
n	20,000	40,000	60,000	80,000	100,000	120,000

TABLE 3
Data Sets to Invesitgate Scalability with Respect to Dimensionality

Data Set	SIM_15	SIM_16	SIM_17	SIM_18	SIM_19	SIM_20
d	20	40	60	80	100	200

TABLE 4
Data Sets to Invesitgate Sensitivity to Number of Points per Clusters

Data Set	SIM_21	SIM_22	SIM_23	SIM_24	SIM_25	SIM_26	SIM_27	SIM_28	SIM_29
y	8	9	10	11	12	13	14	15	16

1. **Data sets to investigate the effect of natural clustering in the data**. Eight simulated data sets (*SIM_1* through *SIM_8*) were generated with eight different values of standard deviation σ to investigate the ability of our algorithm to take advantage of natural clustering in the data. The standard deviation determines how well separated the clusters are: Lower standard deviations ensure nonoverlapping clusters whereas higher standard deviations lead to overlapping clusters. For these data sets, the remaining parameters were fixed at $c = 200$, $d = 10$, $min\_cluster\_size = 10$, and $max\_cluster\_size = 100$. The specific values of the standard deviation used and the exact number of points n in each of these simulated data sets are reported in Table 1.
2. **Data sets to investigate scalability with respect to data set size**. To investigate the impact of data set size on the running time of our algorithm, we created six data sets (*SIM_9* through *SIM_14*) with parameters $c = 200$, $d = 10$, $\sigma = 0.1$, and $min\_cluster\_size = max\_cluster\_size = y$. The number y of points per cluster was increased from 100 to 600 in steps of 100 to create data sets ranging in size n from 10,000 to 120,000 as shown in Table 2.
3. **Data sets to investigate scalability with respect to data dimensionality**. To investigate the running time of our algorithm as the dimensionality of the data d increases, we created six data sets (*SIM_15* through *SIM_20*) each with 10,000 points using parameters $c = 200$, $\sigma = 0.1$, and $min\_cluster\_size = max\_cluster\_size = 50$ The number dimensions for each of these data sets is presented in Table 3.
4. **Data sets to investigate sensitivity to number of points per cluster**. To investigate the sensitivity of our method to the number of points per cluster, we generated nine data sets (*SIM_21* through *SIM_29*) with $min\_cluster\_size = max\_cluster\_size = y$ varying from 8 through 16, using $d = 10$, $\sigma = 0.01$, and $c = 200$, hence, each data set contains $200y$ points as shown in Table 4.

4.2 Microaggregation Methods Used

Statistical agencies typically implement microaggregation by aggregating across a fixed number of records. Further, published results [4] indicate that fixed group size microaggregation results in lower information loss than other heuristic methods. Accordingly, we compare our results with the information loss that results when the fixed group size microaggregation method proposed by Domingo-Ferrer and Mateo-Sanz [4] is used. We refer to their method as *diameter-based* fixed size microaggregation. We also use a *centroid-based* fixed size microaggregation to form fixed size groups. Both of these fixed-size algorithms rely on the process of forming a group G of k points around some data point p. Initially, $G = \{p\}$, and then in each of $k-1$ iterations, we add to G the data point that is closest to G's centroid. The following pseudocode procedure assumes that $p \in S$ and $\mid S \mid \geq k$:

```
gather( Set S, DataPoint p, int k ) {
  G ← {p};
  S ← S - {p};
  loop k-1 times {
    let q ∈ S be closest to the centroid of G;
    G ← G ∪ {q};
    S ← S - {q};
  }
  return G;
}
```

The *diameter-based* fixed-size microaggregation algorithm is from [4]. The idea is to iteratively obtain a pair of data points whose distance is maximal (thereby realizing a diameter of the data set), and then gather a group around each of the two points. If fewer than k points remain, each is added to its nearest group:

```
diameterBasedFixedSize( Set S, int k ) {
  P = ∅;            // P is the partition
  while ( |S| ≥ 2k ) {
    let a and b be a pair of most distant points
      in S;
    P ← P ∪ gather(S, a, k);
    P ← P ∪ gather(S, b, k);
  }
  if ( |S| >= k )
    P ← P ∪ S;
  else // |S| < k
    for each p ∈ S {
      let G ∈ P be the group whose centroid is
        closest to p;
      G ← G ∪ {p};
   }
  return P;
}
```

Instead of forming clusters around the two most distant data points, we propose a variation that forms clusters around a point that is furthest from the centroid. Our *centroid-based* fixed-size microaggregation method is as follows:

```
centroidBasedFixedSize ( Set S, int k ) {
  P = ∅;  // P is the partition
  while ( |S| ≥ k ) {
    let p ∈ S be farthest from the centroid of S;
    P ← P ∪ gather(S, p, k);
  }
  for each p ∈ S {
    let G ∈ P be the group whose centroid is
       closest to p;
    G ← G ∪ {p};
  }
  return P;
}
```

While our MST partitioning algorithm results in groups that contain at least k points, it does not guarantee an upper bound on group size. To reduce information loss, one can further partition groups of size $2k$ and greater. We might consider applying our MST-based method recursively to each such *oversized* group but this is pointless since it invariably leads to the same irreducible tree; for if a different tree were to result, it would have lower cost and so would have replaced the irreducible tree in the original MST. Instead, we use one of the fixed-size methods to partition oversized groups into groups of size less than $2k$. Hence, we arrive at the following microaggregation methods:

- D: Diameter-based fixed size method,
- C: Centroid-based fixed size method,
- M: MST-partitioning algorithm,
- M-d: MST-partitioning followed by clusters of size $\geq 2k$ partitioned by D, and
- M-c: MST-partitioning followed by clusters of size $\geq 2k$ partitioned by C.

4.3 Results

For a given value of minimum group size k, we compare information loss $L = SSE/SST$ (as defined in Section 2 and expressed as a percentage) resulting from alternative microaggregation strategies. Table 5 presents our experimental results. All of the methods ran quickly on the Tarragona, Census, and Creta data sets: the diameter-based method D in under 3 seconds, the centroid-based method C in under 1 second, and the MST partitioning variations M, M-d, and M-c in under 2 seconds. Running times differed on the (much larger) simulated data set $Sim\_1(\sigma = 0.01)$: C ran in about 1 minute, M in 4.2 minutes, and M-d and M-c in about 4.5 minutes. Running time for the diameter-based method D ranged from 44 minutes for $k = 10$ up to 153 minutes for $k = 3$.[2] All programs were written in Java. Tests were performed on a Pentium 4 running Java bytecode at 2.5 Ghz.

For the data sets used in previous studies (Tarragona, Census, and Creta), the information loss obtained under at least one of the fixed size method (D or C) is lower than that obtained using a variation of our proposed methods (M, M-d, or M-c). However, with the simulated data set SIM_1 (with $\sigma = 0.01$), our methods result in significantly lower information loss than fixed-size methods. This is because the simulated data set has pronounced and well separated natural clusters. We investigate the ability of our algorithm to take advantage of natural data clusters using other simulated data sets and summarize our findings in Tables 7 and 8.

Clusters of size $2k$ or greater may result from our MST-partitioning algorithm; Table 6 presents the percentage of all clusters that are oversized, and the average size of these oversized clusters. No more than 16 percent of the clusters formed by MST-partitioning have size $\geq 2k$ and the average size of these oversized clusters is just over $2k$. Since fixed size methods need to be applied to a relatively low proportion of *oversized* clusters, and these clusters are relatively small in size (since k is much smaller than n), the running times for methods M-d and M-c are only marginally higher than that of MST-partitioning alone. Notably, the percentage of oversized clusters is higher for the simulated data set, suggesting that the MST for data possessing pronounced clusters tends to contain nontrivial star-shaped subgraphs (see Appendix B).

Table 7 presents information loss obtained under various methods (for $k = 4$) as the clusters inherent in the data

2. Our implementation of the diameter-based method D uses a stored data approach in which distances between points are computed on-the-fly as needed. The naïve diameter-finding operation is quadratic in the number of points and is performed $n/(2k)$ times on a n-point data set, leading to cubic time complexity. A stored matrix approach, in which distances are precomputed and stored in a matrix, leads to quadratic time complexity but increases the space complexity to $O(n^2)$, making this approach impractical for larger problems such as those represented by our simulated data sets. Notably, the centroid-based method C has quadratic time complexity and linear space complexity, making it more amenable to larger problems.

TABLE 5
Information Loss under Various Methods

100L = 100xSSE/SST	Data Sets	Tarragona	Census	Creta	SIM_1 σ =0.01
k=3	D	15.60	5.42	5.42	0.387
	C	20.74	5.35	5.49	0.366
	M	17.14	6.51	6.51	0.013
	M-d	16.63	6.11	6.11	0.012
	M-c	16.69	6.12	6.16	0.012
k=4	D	19.27	7.22	7.22	0.560
	C	27.80	7.17	7.35	0.552
	M	20.92	8.78	8.78	0.016
	M-d	19.66	8.24	8.25	0.015
	M-c	19.67	8.24	8.23	0.015
k=5	D	22.67	8.81	8.82	0.749
	C	32.47	8.69	8.87	0.717
	M	24.57	10.70	10.70	0.018
	M-d	24.50	10.30	10.28	0.017
	M-c	24.52	10.33	10.31	0.017
k=10	D	36.99	14.55	14.53	1.453
	C	44.90	14.34	14.51	1.430
	M	38.83	17.58	17.58	0.024
	M-d	38.58	17.17	17.12	0.023
	M-c	38.65	17.16	17.17	0.023

TABLE 6
Oversized Clusters (i.e., of size $\geq 2k$) Formed Due to MST-Partitioning

Clusters of Size ≥ 2k		Tarragona	Census	Creta	SIM_1 σ=0.01
k=3	proportion of oversized clusters	10.23%	9.42%	9.42%	13.31%
	average size of oversized clusters	6.27	6.46	6.46	6.66
k=4	proportion of oversized clusters	8.75%	6.22%	6.22%	15.12%
	average size of oversized clusters	8.57	8.38	8.38	9.19
k=5	proportion of oversized clusters	8.13%	10.19%	10.19%	15.24%
	average size of oversized clusters	11.10	12.00	12.00	11.42
k=10	proportion of oversized clusters	9.84%	6.02%	6.02%	15.40%
	average size of oversized clusters	21.50	20.60	20.60	23.13

become more overlapping and less well-defined. Data set *SIM-1* generated with $\sigma = 0.01$ has well separated clusters; the overlap between clusters increases with increasing standard deviation until the clustering effect virtually disappears with $\sigma = 0.3$. The last row of the table (labeled UB) presents an upper bound on information loss for an optimal partition satisfying the group size constraint. Each upper bound was obtained by partitioning each of the 200 simulated clusters—which are known a priori—using the fixed size methods (C and D) and taking the partition that results in the least information loss. The upper bound is tight when clusters are well separated (small σ) and less tight as clusters overlap.

TABLE 7
Effect of Clustering on Information Loss $(k = 4)$

100L	SIM_1	SIM_2	SIM_3	SIM_4	SIM_5	SIM_6	SIM_7	SIM_8
σ	**0.01**	**0.025**	**0.05**	**0.075**	**0.10**	**0.15**	**0.20**	**0.30**
D	0.560	0.584	0.781	1.137	1.685	3.243	5.023	9.069
C	0.552	0.568	0.762	1.146	1.678	3.265	5.065	9.054
M	0.016	0.100	0.390	0.876	1.531	3.418	5.617	10.993
M-d	0.015	0.093	0.359	0.807	1.418	3.154	5.182	10.077
M-c	0.015	0.093	0.362	0.811	1.428	3.179	5.223	10.138
UB	0.014	0.085	0.329	0.738	1.290	2.893	4.779	9.532

Our methods are significantly better with low values of standard deviation $(\sigma < 0.15)$ whereas fixed-size microaggregation methods result in marginally lower information loss when the clustering effect is not pronounced $(\sigma \geq 0.2)$. Fig. 3 highlights this characteristic of our algorithm by presenting the ratio of the information loss obtained using the best of the fixed size methods (C or D) to the best results obtained using one of our methods (M, M-d, or M-c). Notice that when the clusters are well separated $(\sigma = 0.01)$, information loss under a fixed size method (D in this case) is over 37 times that obtained using our method (M-d in this case). For overlapping clusters, on the other hand $(\sigma \geq 0.2)$, though the fixed size methods do marginally better, the ratio of information loss is not significantly less than 1.

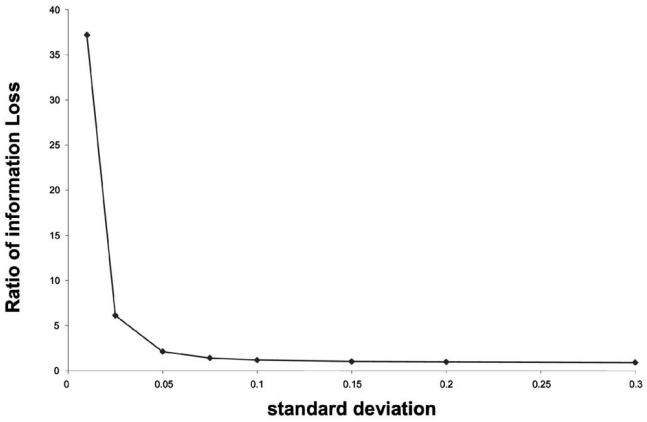


Fig. 3. The ratio of information loss under fixed methods to information loss under MST partitioning approaches 1.0 as inherent clusters become less well separated.

The advantage of our method over fixed-size methods when clusters are well separated obtains whenever the number of points in some of the clusters is not a multiple of k. In such cases, the fixed-size methods are forced to group points belonging to distinct clusters, hence, points that are well separated in space. To investigate this, we ran the methods on simulated data sets *SIM_21* through *SIM_29* each of which contains 200 well-separated clusters of exactly y points per cluster. The results, summarized in Table 8, indicate that the MST-based methods are comparable to the fixed-size methods when y is a multiple of k, but result in significantly lower information loss otherwise. The last row of Table 8 presents the ratio of information loss of the best result obtained from a fixed-size method (D or C) to that obtained from an MST-based method (M-d or M-c). In practical applications, the number of points per cluster can be expected to vary from cluster to cluster, and there will exist no value k of which every cluster size will be a multiple.

We next examine the impact of data set size on the running time of our algorithm using simulated data sets (*SIM_9* through *SIM_14*) ranging in size from 10,000 to 120,000 points. Table 9 presents the time taken for each phase of our algorithm. Although both Prim's algorithm and the edge cutting phase exhibit the quadratic time behavior indicated by our analysis in Section 3—doubling the problem size results in a quadrupling of running time—the constant factors are very different. Total running time is clearly dominated by Prim's algorithm. For large problems, one can build the MST once and then partition it for different values of k, thereby amortizing the cost of MST construction.

Using data sets *SIM_15* to *SIM_20*, each of which contains 10,000 data points, we then investigated the impact of the number of dimensions on the running time of MST partitioning. As Table 10 indicates, the dimensionality of the data has little impact on the total running time.

5 Conclusions

Our experimental results indicate that with the Tarragona, Census, and Creta data sets, the fixed group size methods

TABLE 8
Effect of Fixed Number y of Points Per Cluster $(k = 4)$.

	SIM_21	SIM_22	SIM_23	SIM_24	SIM_25	SIM_26	SIM_27	SIM_28	SIM_29
y	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
D	0.022	4.195	3.506	3.556	0.020	3.067	2.679	2.575	0.018
C	0.022	4.091	3.650	3.536	0.020	2.946	2.636	2.527	0.018
M	0.024	0.022	0.022	0.022	0.022	0.021	0.021	0.020	0.020
M-d	0.021	0.020	0.021	0.021	0.020	0.020	0.019	0.019	0.019
M-c	0.021	0.020	0.021	0.021	0.020	0.020	0.019	0.019	0.019
LossFixed/ LossMST	1.048	204.550	166.952	168.381	1.000	147.300	138.737	133.000	0.947

TABLE 9
Running Time with Increasing Data Set Size $(k = 4)$

	SIM_9	SIM_10	SIM_11	SIM_12	SIM_13	SIM_14
Total number of points	**20000**	**40000**	**60000**	**80000**	**100000**	**120000**
Total time (all times in seconds)	**668**	**3819**	**7505**	**14374**	**23785**	**33894**
Prim's method (phase 1A)	661	3793	7448	14271	23631	33665
Descendant count initialization (phase 1B)	0.02	0.03	0.03	0.05	0.05	0.06
Edge cutting (phase 2)	7	25	56	102	153	227
Cluster formation (phase 3)	0.06	0.03	0.13	0.20	0.22	0.16

TABLE 10
Running Time with Increasing Number of Dimensions $(k = 4)$

	SIM_15	SIM_16	SIM_17	SIM_18	SIM_19	SIM_20
Dimension	**20**	**40**	**60**	**80**	**100**	**200**
Time (seconds)	123	129	130	165	157	207

TABLE 11
The Number of $[k, 2k-1]$-Partitions of n Data Points

n	k=3	k=4	k=5	k=10
10	2226	336	126	1
15	10461451	981981	137566	1
20	182285304201	13519012221	790940436	92378
50	1.425e43	4.529e41	3.429e39	4.245e29
100	7.697e108	4.425e107	2.900e104	2.066e87
200	1.163e263	8.639e263	3.621e259	1.094e230

result in marginally lower information loss than the methods based on our MST-partitioning algorithm. However, with simulated data that possess well-separated clusters, the MST-based partitioning algorithm does significantly better than fixed size approaches. The more pronounced the inherent clustering effects in the data (as characterized by low within-group variances relative to between group variances), the greater is the advantage of using our methods. This suggests that our MST partitioning-based method should be considered as a potential candidate for any practical application. Our method exploits natural clustering effects in the data consistent with the objective of minimum information loss.

The performances of the two fixed group size methods are similar. This suggests that for fixed size microaggregation, our centroid-based method should be tried along with the diameter-based fixed size method proposed by Domingo-Ferrer [4]. Furthermore, each of these fixed size methods should be used to partition groups of size $\geq 2k$ that result from MST partitioning. Experimental results indicate that only a relatively small percentage of the groups formed by MST partitioning exceed the maximum group size of $2k-1$. Moreover, since these oversized groups tend to be small (of size not much greater than $2k$), the computational overhead involved in further partitioning of these groups is not significant.

Our work has demonstrated the intractability of the microaggregation problem and the need for heuristics. We have also presented an MST-based algorithm for this problem and compared this algorithm to other heuristics. Although work on this particular version of the clustering problem is relatively recent, clustering, in general, has been researched extensively for decades. One avenue of future work is to adapt some of the ideas used to solve other clustering problems to this constrained version. Another line of work is to explore methods where minimum group size is treated as a soft constraint associated with a preference level.

APPENDIX A
A RESULT ON SEARCH SPACE SIZE

We use recurrence relations to characterize the number of partitions whose groups have size between k and $2k-1$ inclusive. Let us say that a $[k, k_0]$-partition is a partition whose groups have a size between k and k_0 inclusive. Then, where $P^{k_0}(k,n)$ denotes the number of $[k, k_0]$-partitions of n data points, we have:

$$P^{k_0}(k,n) = \begin{cases} P_0^{k_0}(k,n) & \text{if } k \leq n \text{ and } k \leq k_0 \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

The function $P_m^{k_0}(k,n)$ denotes the number of $[k, k_0]$-partitions of n points that contain at least m groups of size k. We then have:

$$P_m^{k_0}(k,n) = \begin{cases} 0 & \text{if } n < mk \\ G_m(k,n) & \text{if } n = mk \\ P^{k_0}(k+1, n-mk) \cdot G_m(k,n) + P_{m+1}^{k_0}(k,n) & \text{otherwise.} \end{cases} \quad (5)$$

Here, $G_m(k,n)$ represents the number of ways of dividing n points into $m > 0$ groups of size k and an additional group of size $n - mk$, where the order of the m groups is immaterial. Consider the three cases represented in (5). First, if mk exceeds n, the value of $P_m^{k_0}(k,n)$ is zero since a partition of n points into m groups of size k is not possible. Second, if mk is equal to n, the number of different ways of dividing n points into m groups of size k is precisely $G_m(k,n)$. The third case says that the number of partitions is equal to the number of partitions containing exactly m groups of size k (in addition to any number of larger groups), plus the number of partitions containing strictly greater than m groups of size k (again, in addition to any number of larger groups).

For completeness, we note that:

$$\begin{aligned} G_m(k,n) &= \frac{\binom{n}{k} \cdot \binom{n-k}{k} \cdot \binom{n-2k}{k} \cdot \ldots \cdot \binom{n-(m-1)k}{k}}{m!} \\ &= \frac{n!}{(k!)^m (n-mk)! m!}. \end{aligned} \quad (6)$$

See Table 11 to gain a feel for how fast the function $P^{2k-1}(k,n)$ grows as n increases.

APPENDIX B
ON IRREDUCIBLE TREES

A tree T is *irreducible* if it contains no removable edges, that is, if it contains no edge e such that each of the two trees in the graph $T - \{e\}$ has size at least k. The edge-cutting phase of our algorithm produces a forest of irreducible trees each of which corresponds to a cluster. In this appendix, we characterize the structure of irreducible trees.

We refer to a tree as *star-shaped* if it contains some vertex v such that each of the trees to which v is connected by an edge has size strictly less than k. Vertex v is called a *center* of the tree.

Theorem. *A tree is irreducible if and only if it is star-shaped.*

We first assume that tree T is star-shaped and show that no edge of T is removable. Suppose vertex $c \in T$ is a center. If edge e is incident to c, edge e is also incident to a tree of size less than k, implying that e is not removable. Alternatively, if edge e is not incident to c, it belongs to some tree of size less than k that is connected to center c. Deleting e results in two trees—a tree of size less than k and a second tree that connects to c—implying that e is not removable. Since no edge of T is removable, T is irreducible.

To show the other direction, we use induction on tree size to show that every irreducible tree is star-shaped. For the basis, note that every tree of size no greater than k is star-shaped—every vertex is a center—so, every irreducible tree of size no greater than k is star-shaped. For the inductive step, suppose that T is an irreducible tree of size $n > k$. Let v be some leaf node of T. Since T is irreducible, the tree $T' = T - \{v\}$ is irreducible and, therefore, by the inductive hypothesis, T' is star-shaped. Consider the unique path from vertex v to some center $c \in T'$. Let the edge $e = (w, c)$ be the last edge along this path, where it is possible that $v = w$. Let T_w and T_c be the two trees that result from deleting edge e from T, where $v, w \in T_w$ and $c \in T_c$. Since T is irreducible, two cases are possible:

Case 1. $|T_w| < k$. Every tree connected to vertex c has a size less than k, implying that c is a center in tree T and, thus, T is star-shaped.

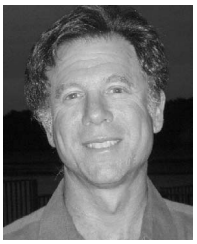
Case 2. $|T_c| < k$. We argue that vertex w is a center in tree T. If $w = v$, then w is connected to only the tree T_c whose size is less than k, implying that w is a center and that T is star-shaped. Alternatively, suppose $v \neq w$. Since c is a center in tree T', we know that $|T_w - \{v\}| < k$ and, hence, $|T_w| \leq k$. Since no more than k vertices comprise T_w, each of the trees to which w is connected in T_w has size less than k. Moreover, since T_c, the only remaining tree to which w is connected, has size less than k, w must be a center in tree T and, so, T is star-shaped.

ACKNOWLEDGMENTS

The authors thank Josep Domingo-Ferrer for providing the Tarragona, Census, and Creta data sets. They also thank the anonymous reviewers for their helpful suggestions.

REFERENCES

[1] N.R. Adam and J.C. Wortmann, "Security Control Methods for Statistical Databases: A Comparative Study," *ACM Computing Surveys,* vol. 21, no. 4, pp. 515-556, 1989.
[2] T. Cormen, C. Leiserson, and R Rivest, *Introduction to Algorithms,* second ed. Cambridge: McGraw Hill, 2001.
[3] R.A. Dandekar, J. Domingo-Ferrer, and F. Sebe, "LHS-Based Hybrid Microdata vs. Rank Swapping and Microaggregation for Numeric Microdata Protection," *Inference Control in Statistical Databases, Lecture Notes in Computer Science 2316,* J. Domingo-Ferrer, ed., Springer-Verlag, 2002.
[4] J. Domingo-Ferrer and J.M. Mateo-Sanz, "Practical Data-Oriented Microaggregation for Statistical Disclosure Control," *IEEE Trans. Knowledge and Data Eng.,* vol. 14, no. 1, pp. 189-201, Jan./Feb. 2002.
[5] J. Domingo-Ferrer and V. Torra, "A Quantitative Comparison of Disclosure Control Methods for Microdata," *Confidentiality, Disclosure, and Data Access: Theory and Practical Application for Statistical Agencies,* P. Doyle, J. Lane, J. Theeuwes, and L. Zayatz, eds., pp. 111-133, Amsterdam: North-Holland, 2001.
[6] S.L. Hansen and S. Mukherjee, "A Polynomial Algorithm for Optimal Univariate Microaggregation," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 4, pp. 1043-1044, July/Aug. 2003.
[7] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys,* vol. 31, no. 3, 1999.
[8] T. Kanungo, D.M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A.Y. Wu, "An Efficient k-Means Clustering Algorithm: Analysis and Implementation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 24, pp. 881-892, 2002.
[9] A. Oganian and J. Domingo-Ferrer, "On the Complexity of Optimal Microaggregation for Statistical Disclosure Control," *Statistical J. United Nations Economic Commission for Europe,* vol. 18, no. 4, pp. 345-354, 2001.
[10] R. Sedgewick, *Algorithms,* second ed., Addison-Wesley, 1988.
[11] J.H. Ward, "Hierarchical Grouping to Maximize an Objective Function," *J. Am. Statistical Assoc.,* vol. 58, pp. 236-244, 1963.
[12] A. Willenborg and T. De Waal, *Elements of Statistical Disclosure Control.* New York: Springer-Verlag, 2000.
[13] C.T. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," *IEEE Trans. Computers,* vol. 20, no. 4, pp. 68-86, Apr. 1971.



Michael Laszlo received the PhD degree in computer science from Princeton University in 1987. He is a professor in the Graduate School of Computer and Information Sciences at Nova Southeastern University. He is the author of several textbooks on programming and computer graphics, and his work focuses on computer graphics, algorithms, and programming.

Sumitra Mukherjee received the PhD degree in decision and information systems from Carnegie Mellon University in 1994. He is a professor in the Graduate School of Computer and Information Sciences at Nova Southeastern University. His research interests include data security and artificial intelligence. He has been published in the *IEEE Transactions on Knowledge and Data Engineering*, the *Management Science*, the *Journal of the American Statistical Association*, and the *Operations Research Letters* among other journals. He is a member of the IEEE.