Clustering with Minimum Spanning Trees

Yan Zhou [a], Oleksandr Grygorash [b], Thomas F. Hain [a], Meador Inge [a], Zach Jorgensen [a]

[a]*School of Computer and Information Sciences*
University of South Alabama, Mobile, AL 36688 USA

[b]*Urban Insight Inc.*
5657 Wilshire Blvd Ste 290, Los Angeles, CA 90036

Abstract

We propose two Euclidean minimum spanning tree based clustering algorithms — one a k-*constrained*, and the other an *unconstrained* algorithm. Our k-constrained clustering algorithm produces a k-partition of a set of points for any given k. The algorithm constructs a minimum spanning tree of a set of representative points and removes edges that satisfy a predefined criterion. The process is repeated until k clusters are produced. Our unconstrained clustering algorithm partitions a point set into a group of clusters by maximally reducing the overall standard deviation of the edges in the Euclidean minimum spanning tree constructed from a given point set, without prescribing the number of clusters. We present our experimental results comparing our proposed algorithms with k-means and the Expectation-Maximization (EM) algorithm on both artificial data and benchmark data from the UCI repository. We also apply our algorithms to image color clustering and compare them with the standard minimum spanning tree clustering algorithm.

Key words: Minimum spanning trees, k-constrained clustering, unconstrained clustering, representative point sets, standard deviation reduction

1 Introduction

Clustering algorithms for point sets in a metric space ($\mathbb{E}^d$, where d is the number of dimensions) are often based on Euclidean Minimum Spanning Trees (EMST), in

Email addresses: `zhou@cis.usouthal.edu` (Yan Zhou), `ogrygorash@gmail.com` (Oleksandr Grygorash), `thain@usouthal.edu` (Thomas F. Hain), `wmi601@usouthal.edu` (Meador Inge), `zdjorgen@usouthal.edu` (Zach Jorgensen).

which the weight of an edge is the Euclidean distance between its incident points. The cost of constructing an EMST is $O(n^2 \log n)$, where n is the number of points. Such algorithms are known to be capable of detecting clusters with irregular boundaries. In these algorithms, the points in the metric space are partitioned by the strategic removal of edges of the EMST, generating subtrees, each of which represents a cluster.

The inherent cluster structure of a point set is closely related to the objects and/or concepts that are embedded within that set. In practice, there are two general types of clustering problems. In the first type, the number of embedded objects can be acquired with the help of application domain experts. Here the input to an algorithm will specify (in addition to the point set) the number of clusters, k, to be formed. In the current context, we will call these *k-constrained algorithms*. In the second type of problem, information on the number of embedded objects is hidden, and thereby unavailable as an input to the clustering algorithm. We term such algorithms *unconstrained algorithms*. In addition, some algorithms may also include as part of their input a small number of tuning parameters whose values depend on object characteristics.

An example application of EMST-based algorithms, and motivating the current work, is color clustering in web image analysis. Analyzing such images—for example in preparation for the extraction of textual content—may be complicated by the image having complex backgrounds, and often many colors. In this domain, colors are represented as points in a three-dimensional (e.g., RGB, or HSV) color space.

In this paper, we present two EMST-based algorithms—one a k-constrained, and the other an unconstrained algorithm—which address some of the shortcomings of existing clustering algorithms, such as poor cluster discrimination, or (in the case of unconstrained algorithms) the generation of an excessive number of clusters.

In Section 2, we review the existing EMST-based clustering algorithms, and other related work. Sections 3 and 4 present respectively our k-constrained and our unconstrained algorithm. In Section 5, we provide experimental results comparing our algorithms with existing ones. Section 6 provides conclusions from our work, and discusses future directions.

2 Related Work

The simplest k-constrained EMST-based algorithm is to remove $k - 1$ edges from the EMST, resulting in k subtrees. Each cluster is the set of points in each subtree. In the remaining paper, we will refer to this algorithm as SEMST.

In the mid 80's, Avis [2] found an $O(n^2 \log^2 n)$ algorithm for the min-max diameter 2 clustering problem. Asano, Bhattacharya, Keil, and Yao [1] later gave an optimal $O(n \log n)$ algorithm using maximum spanning trees for minimizing the maximum diameter of a bipartition. The problem becomes NP-complete when the number of partitions is beyond two [9]. Asano also considered the clustering problems in which the goal is to maximize the minimum intercluster distance. They gave an $O(n \log n)$ algorithm for computing a k-partition of a point set by removing the $k - 1$ longest edges from the minimum spanning tree constructed from that point set [1].

Zahn [16] describes a method to remove *inconsistent* edges—edges, whose weights are significantly larger than the average weight of nearby edges—from the EMST. His definition of inconsistent edges relies on the concept of *depth-d neighborhoods*, N_1 and N_2, for each incident point, v_1 and v_2, of an edge e. The neighborhood of v_1 is the set of edges on paths from v_1 having length no greater than d, and excluding the edge e. Let $\overline{w}_{N1}$ be the average weight, and σ_{N1} be the standard deviation, of neighborhood N_1, with similar definitions for N_2. He provides several alternative inconsistency criteria:

(1) $w > \overline{w}_{N1} + c \times \sigma_{N1}$
(2) $w > max(\overline{w}_{N1} + c \times \sigma_{N1}, \overline{w}_{N2} + c \times \sigma_{N2})$
(3) $\dfrac{w}{max(\overline{w}_{N1}, \overline{w}_{N2})} > f$

The values d, c and f are user-assigned tuning parameters. Each cluster contains the points within each subtree resulting from the removal of inconsistent edges from the EMST. We will denote this algorithm as ZEMST.

Eldershaw and Hegland [6] re-examine the limitations of many 2D clustering algorithms that assume that clusters of a point set are essentially spherical, and provide a broader definition of a cluster based on transitivity: if two points p_1 and p_2 are close to the same point p_0, they are both members of the same cluster. They present an algorithm which constructs a graph using Delaunay triangulation, and remove edges that are longer than a cut-off point. Next, they apply a graph partitioning algorithm to find the isolated connected components in the graph, and each discovered component is treated as a cluster. Unlike Zahn's method in which inconsistency is a locally determined property of an edge, they choose a cut-off point which corresponds to a global minimum.

Bansal, Blum and Chawla [3] introduced an unconstrained algorithm called correlation clustering. The clustering problem they consider is a complete graph in which each edge is labeled qualitatively as "+" (similar) or "-" (dissimilar). The objective is to find the clustering that minimizes the number of disagreements with the edge labels. They proved NP-hardness of the fundamental problem, but povided approximation algorithms which have been further improved by others [5,4].

More recently, Päivinen [12] proposed a scale-free minimum spanning tree clustering algorithm which constructs a scale-free network and outputs clusters containing highly connected vertices. We will refer to this algorithm as SFMST.

Xu Ying, Olman and Xu Dong [14] use an EMST-based algorithm to represent multidimensional gene expression data. They point out that an EMST-based clustering algorithm does not have to assume that points within a cluster are grouped around centers or separated by a regular geometric curve. They describe three objective functions, and corresponding k-constrained algorithms. The first objective function constitutes the implementation of SEMST. The second objective function is defined to minimize the total distance between the center and each data point in a cluster. This algorithm first removes $k-1$ edges from the tree, creating a k-partition. Next, it repeatedly merges a pair of adjacent partitions and finds its optimal 2-clustering solution. They observe that the algorithm quickly converges to a local minimum. The third objective function is defined to minimize the total distance between a $representative$ point of a cluster and each other point in the cluster. The representatives are selected so that the objective function is optimized. This algorithm has exponential worst-case running time.

Xu and Uberbacher [15] partition a gray-level image into homogeneous regions by constructing an MST from the image. The tree partitioning algorithm minimizes the sum of the variations of the gray-levels of all subtrees, and the gray-levels of two adjacent subtrees are required to be significantly different. Each subtree contains several gray-levels and represents a homogeneous region in the image. Other applications of the MST clustering algorithm in the area of image processing can be found in [13,7,10]. Lopresti and Zhou [10] suggest an EMST-based color clustering method where each distinct color is a point in the metric RGB color space. They point out that such algorithms may fail when dealing with textures/dithering, or when there is a very large number of colors in an image.

3 Hierarchical EMST-based Algorithm (HEMST)

The simple heuristic used in some k-constrained EMST clustering algorithms, i.e. removing $k-1$ longest edges, often does not work well. Figure 1 [16] illustrates a typical example of the cases in which simply removing the $k-1$ longest edges does not necessarily output the desired cluster structure. This is also a good example where the nearest neighbor or the single linkage [8] method does not work well.

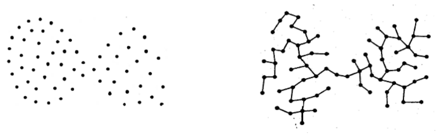


Fig. 1. Clusters connected through a point.

We now present a more effective k-constrained algorithm, which is a hierarchically derived EMST-based clustering algorithm on a point set. We will call it HEMST from now on.

To simplify discussion, in the current context we will refer to an EMST simply as *tree*. It can be noted that edge removal from such a tree is a closed operation on EMSTs since each of the resulting subtrees is also an EMST.

In order to explain the algorithm, we now introduce the notion of a representative point of a cluster. This is a point in the cluster closest to the geometric centroid of that cluster. For the sake of algorithmic orthogonality, the points in the input point set can be considered as representative points, each representing only themselves, and thus the data contained in tree nodes are representative points. A representative point has associated with it the set of representative points that it represents. The weight of a tree edge is the Euclidean distance between its incident representative points. We furthermore will omit the word *representative*, when the context is clear.

We will define the following functions in our algorithmic description of HEMST. All functions returning a set are bolded.

$CreateEMST(\mathbf{S})$	generates an (EMS) tree from point set **S**
$\mathbf{EdgeSet}(T)$	returns the set of edges in tree T
$\mathbf{Points}(T)$	returns the set of [representative] points in tree T
$AvgWeight(T)$	returns the average weight of edges in tree T
$StdDev(T)$	returns the standard deviation of weights of edges in tree T
$Centroid(T)$	returns the centroid of the [representative] points in tree T
$\mathbf{ParEMST}(T, \mathbf{E})$	returns the partition (set of subtrees) resulting from the removal of edges in set **E** from tree T
$D(p_1, p_2)$	returns the Euclidean distance between points p_1 and p_2
$ReprPoint(T)$	returns the [representative] point in tree T, being the point in T that is closest to the centroid of all its points, i.e., $p_i \in \{T \mid D(p_i, Centroid(T)) = \min_{p_j \in T} D(p_j, Centroid(T))\}$
$\mathbf{PointSet}(p)$	recursive function generating the original points (in **S**) represented directly or indirectly by a [representative] point, p. $\mathbf{PointSet}(p) = \begin{cases} p & \text{if } p \in \mathbf{S}, \\ \bigcup_{p_i \in \mathbf{PointSet}(p)} \mathbf{PointSet}(p_i) & \text{otherwise.} \end{cases}$

Let the input point set for the algorithm be $\mathbf{S}$. The tree $T(\mathbf{S})$ is initially partitioned in an unconstrained manner by removing all edges whose weight $w >$

$AvgWeight(T) + StdDev(T)$, resulting in a set of subtrees $\mathbf{T} = \{T_1, T_2, ...\}$, each representing an initial cluster. If $|\mathbf{T}| \le k$ then $k - |\mathbf{T}|$ additional edges having greatest weight are removed from $T(\mathbf{S})$, generating k subtrees, and k corresponding output clusters. If $|\mathbf{T}| > k$, the centroid of each cluster is used to find a representative point p_i for that cluster, i.e., $p_i = ReprPoint(T_i)$. At this stage, this set of representative points forms a new point set $\mathbf{S}' = \{p_i | p_i = ReprPoint(T_i)\}$. The process is recursively repeated until the number of clusters is k. Finally, the original point set belonging to each of the k clusters is recursively generated by the union of the point sets associated with each representative point in the cluster. The pseudo-code for HEMST is given in Algorithm 1.

```
Function HEMST($\mathbf{S}$,$k$)
Input: Point set $\mathbf{S}$, and required number of clusters, $k$
Output: Set of $k$ point sets (set of clusters), $\mathbf{S_k}$
$T \leftarrow$ CreateEMST($\mathbf{S}$);
$\mathbf{S_K} \leftarrow \emptyset$;
$\overline{w} \leftarrow$ AvgWeight($T$);
$\sigma \leftarrow$ StdDev($T$);
$\mathbf{E} \leftarrow \emptyset$ ;                         /* $\mathbf{E}$ is a set of edges */
forall $e \in$ EdgeSet($T$) do
    if Weight($e$) $> \overline{w} + \sigma$ then
        $\mathbf{E} \leftarrow \mathbf{E} \cup \{e\}$;
    end
end
$\mathbf{T_k} \leftarrow$ ParEMST($T, \mathbf{E}$) ;          /* $\mathbf{T_k}$ is a set of subtrees */
if $|\mathbf{T_k}| > k$ then
    $\mathbf{P} \leftarrow \emptyset$ ;     /* $\mathbf{P}$ is a set of representative points */
    forall $T \in \mathbf{T_k}$ do
        $\mathbf{P} \leftarrow \mathbf{P} \cup \{ReprPoint(T)\}$;
    end
    HEMST($\mathbf{P}, k$) ;                 /* Recursive call to HEMST */
else
    /* $\mathbf{PQ}$ is a priority queue of edges in $\mathbf{T_k}$ in order
    of weights                                                  */
    $\mathbf{PQ} \leftarrow$ PriorityQueue($\mathbf{T_k}$);
    for $i \leftarrow 1$ to $k - |\mathbf{T_K}|$ do
        $\mathbf{E} \leftarrow \mathbf{E} \cup \{$Dequeue($\mathbf{PQ}$)$\}$;
    end
    $\mathbf{T_k} \leftarrow$ ParEMST($T, \mathbf{E}$);
    forall $T \in \mathbf{T_k}$ do
        $\mathbf{S_k} \leftarrow \mathbf{S_k} \cup \{$PointSet(ReprPoint($T$))$\}$;
    end
    return $\mathbf{S_k}$;
end
```

Algorithm 1: HEMST

It might be noted that, in a typical scenario, $k \ll |\mathbf{S}|$. The number of edges whose weights are greater than the mean by one standard deviation (assuming a normal distribution, $\mathbb{Z}(x)$, of edge weights) is $1 - \mathbb{Z}(1) \approx 0.16$ (or slightly less than one in six). Under this assumption, each representative point will represent about six points, and the total number of recursion iterations will be in the order of $\log_6(|\mathbf{S}|/k)$.

Given that each iteration requires an amount of work bounded by $O(n^2 \log n)$ (finding the minimum spanning tree for the representative points, where n is the number of points in the (sub)tree), the algorithm has a complexity given by the recurrence

$$\begin{cases} T_n = 6T_{n/6} + n^2 \log_2 n \\ T_k = 1 \end{cases}$$

yielding a closed form complexity for HEMST of $O(n^2 \log n)$.

Given the same input as shown in Figure 1, our HEMST algorithm, on the other hand, will find the appropriate representative points for the desired clusters. Figure 2 shows the possible distribution of the representative points after a few rounds. Eventually, if given $k = 2$, our algorithm will detect the two-cluster structure.

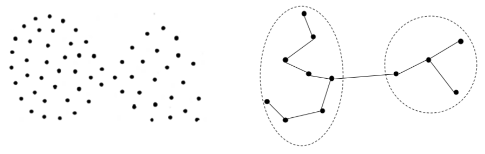


Fig. 2. The representative points of the two clusters connected through a point.

4 Maximum Standard Deviation Reduction Algorithm (MSDR)

The algorithm described next is an EMST-based unconstrained algorithm which tries to discover the underlying cluster structure in the input point set. It is based, unlike correlation clustering introduced by Bansal et al. [3], on removing tree edges that contribute the most to the tree's overall standard deviation of edge weights. The underlying idea is that those edges will tend to be inter-cluster rather than intra-cluster.

4.1 Our MSDR Clustering Algorithm

The algorithm first builds an EMST from the point set $\mathbf{S}$, and removes edges from this tree one-by-one, partioning the zero-generation tree into successive generations of sets of subtrees. After i $(i \geq 0)$ edges removals (i.e. i^{th} generation), the partition (set of subtrees) is denoted $\mathbf{T}^{(i)} = \{T_1^{(i)}, T_2^{(i)}, ..., T_k^{(i)}\}$. Let $\sigma(\mathbf{T}^{(i)})$ represent the average standard deviation of the i^{th} generation partition, defined as,

$$\sigma(\mathbf{T}^{(i)}) = \frac{\sum_{j=1}^{|\mathbf{T}^{(i)}|} |T_j^{(i)}| \cdot \sigma(T_j^{(i)})}{\sum_{j=1}^{|\mathbf{T}^{(i)}|} |T_j^{(i)}|}$$

where $|T_j^{(i)}|$ denotes the number of edges in the j^{th} subtree of that generation. The standard deviation of the partition is the weighted average of the standard deviations of the weights of the edges in each subtree, weighted by the number of its edges.

The edge to be removed from one of the subtree in $\mathbf{T}^{(i-1)}$, is the one that will maximize the reduction of the partition standard deviation, in the i^{th} generation, i.e., $\mathbf{T}^{(i)} = \arg\max(\Delta\sigma(\mathbf{T}^{(i)}))$, where $\Delta\sigma(\mathbf{T}^{(i)}) = \sigma(\mathbf{T}^{(0)}) - \sigma(\mathbf{T}^{(i)})$. The iterative edge removal process stops when $|\Delta\sigma(\mathbf{T}^{(i)}) - \Delta\sigma(\mathbf{T}^{(i-1)})| < |\epsilon \cdot (\Delta\sigma(\mathbf{T}^{(i)}) + 1)|$.

At this time, a 6^{th} order regression is performed on the successive values of $\Delta\sigma(\mathbf{T}^{(i)})$ as a function of i, and the number of clusters k chosen as appropriate for the input point set is taken as $k = \lfloor i' \rfloor$, where i' is the value of i where the regression curve has its first minimum. We find that, in practice, such a point always exists. The final clusters are formed from the points of the subtrees in the set $\mathbf{T}^{(k)}$.

A function $ReduceStdDev(\mathbf{T}, T, e)$ is defined to take a partitioned tree $\mathbf{T}$ (i.e., a set of subtrees), a subtree T within that set, and an edge e within that subtree, and return a tuple, being the new partition after e is removed from T, and the standard deviation reduction of the new partition.

The algorithm pseudo-code is given in Algorithm 2.

4.2 Supporting Data Structure

Whereas an EMST is conceptually an undirected tree, the implementation uses a directed tree whose hierarchy is established (somewhat arbitrarily) during its building phase. Each node has a collection of child nodes, and (with the exception of the root) embeds information about its [parent] edge. Each node is therefore the root of a subtree, and stores information about that subtree. In particular, it stores the number of nodes, the sum of the edge weights, and the sum of the squares of the edge weights in the subtree. All these attributes are efficiently (i.e., in linear time) gathered during the tree building phase. This allows the standard deviations of the edge weight of the subtrees resulting from an edge removal (required by the MSDR algorithm) to be calculated efficiently.

Removing edge e from tree T_0, results in its being partitioned into two subtrees, T_1

```
Function MSDR(S)
Input: Point set $\mathbf{S}$
Output: Set of $k$ point sets (set of clusters), $\mathbf{S_k}$
$T \leftarrow$ CreateEMST($\mathbf{S}$);
$\mathbf{T}^{(0)} \leftarrow \{T\}$;
$i \leftarrow 0$;
repeat
    forall $T \in \mathbf{T}^{(i)}$ do
        $\Delta\sigma_{max} \leftarrow 0$;
        forall $e \in$ EdgeSet$(T)$ do
            $(\mathbf{T_{temp}}, \Delta\sigma_{temp}) \leftarrow$ ReduceStdDev $(\mathbf{T}^{(i)}, T, e)$;
            if $\Delta\sigma_{temp} > \Delta\sigma_{max}$ then
                $\Delta\sigma_{max} \leftarrow \Delta\sigma_{temp}$;
                $\mathbf{T}^{(i+1)} \leftarrow \mathbf{T_{temp}}$;
            end
        end
    end
    $i \leftarrow i + 1$;
until $\begin{cases} \sigma(\mathbf{T}^{(0)}) < \sigma(\mathbf{T}^{(1)}) & \texttt{for } i = 1 \\ \left|\Delta\sigma(\mathbf{T}^{(i)}) - \Delta\sigma(\mathbf{T}^{(i-1)})\right| < \left|\epsilon \cdot (\Delta\sigma(\mathbf{T}^{(i)}) + 1)\right| & \texttt{for } i > 1 \end{cases}$;
$f \leftarrow$ PolyRegression$(\bigcup_{j=1}^{i-1} \Delta\sigma(\mathbf{T}^{(j)}))$;
$k \leftarrow \min(j \in [1, i-1] | f'(j) = 0 \ \& \ f''(j) > 0)$;
$\mathbf{S_k} \leftarrow \emptyset$;
forall $T \in \mathbf{T}^{(k)}$ do
    $\mathbf{S_k} \leftarrow \mathbf{S_k} \cup \{$Points$(T)\}$;
end
return $\mathbf{S_K}$;
```

Algorithm 2: MSDR

and T_2, as seen in Figure 3.

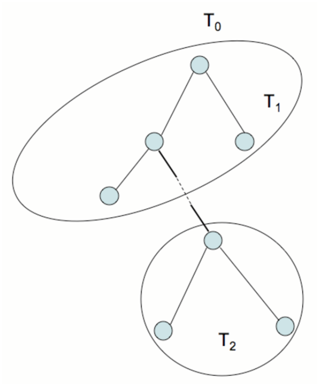


Fig. 3. T_0 is split into T_1 and T_2.

Based on the following properties:

$$\sum_{e_i \in T_1} e_i = \sum_{e_j \in T_0} e_j - \sum_{e_k \in T_2} e_k - e_r$$

$$\sum_{e_i \in T_1} e_i^2 = \sum_{e_j \in T_0} e_j^2 - \sum_{e_k \in T_2} e_k^2 - e_r^2$$

$$\mid T_1 \mid = \mid T_0 \mid - \mid T_2 \mid -1$$

the standard deviations of T_1 and T_2 can be calculated in the following manner,

$$\sigma(T_1) = \frac{\sum_{e_i \in T_1} {e_i}^2}{\mid T_1 \mid} - \left(\frac{\sum_{e_i \in T_1} e_i}{\mid T_1 \mid} \right)^2$$

$$\sigma(T_2) = \frac{\sum_{e_i \in T_2} {e_i}^2}{\mid T_2 \mid} - \left(\frac{\sum_{e_i \in T_2} e_i}{\mid T_2 \mid} \right)^2$$

Since all the summations have been pre-computed, the calculation of both standard deviations can be done in constant time. Considering this fact, and the algorithm outlined in the pseudo-code, the complexity of the algorithm is $O(n^2 \log n + nk)$, where n is the cardinality of the point set, and k is the number of clusters generated. The only assumption is that the minimum of the regression $(\sim k)$ is reasonably porportional to the number of iterations actually performed (i.e. generations generated). The first term in the complexity is from the building of the EMST.

5 Experimental Results

We performed three experiments to demonstrate the effectiveness of our proposed clustering algorithms. In the first experiment, we selected three clustering problems and compared the two proposed algorithms to k-means and EM respectively. In our second experiment, we compared our proposed algorithms to the standard EMST based algorithms—SEMST and ZEMST in image color clustering. In the third experiment we tested our unconstrained MSDR algorithm with datasets from the UCI repository [11].

5.1 HEMST, k-Means, MSDR, and EM

We selected three relatively difficult clustering problems in this experiment. The first problem is presented in Figure 4, in which two clusters are desired. Each cluster is formed as a curving irregular shaped line. The second problem shown in Figure 5 contains two clusters, with one inside the other. The third problem shown in Figure 6 contains two clusters, each with a non-homogeneous density.

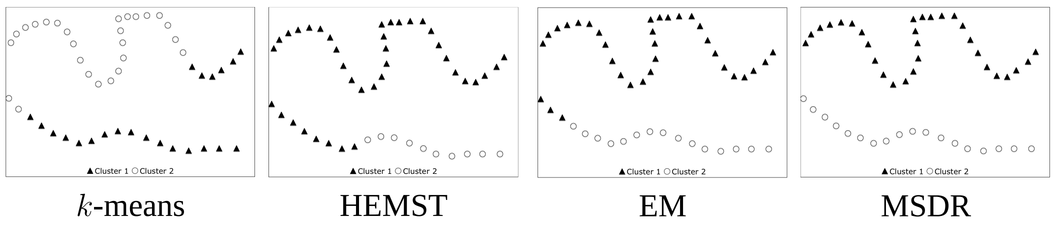


Fig. 4. Two clusters formed by two lines.

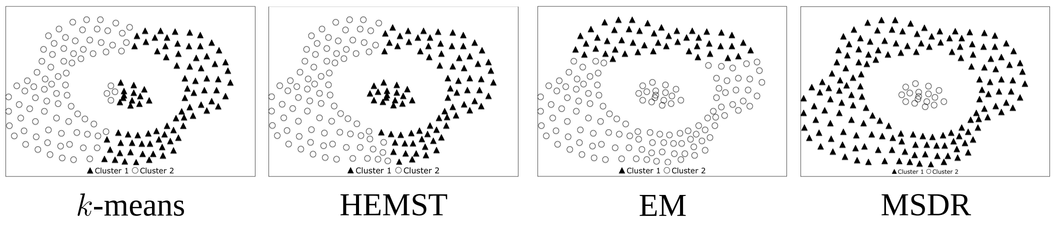


Fig. 5. Two clusters—one inside the other.

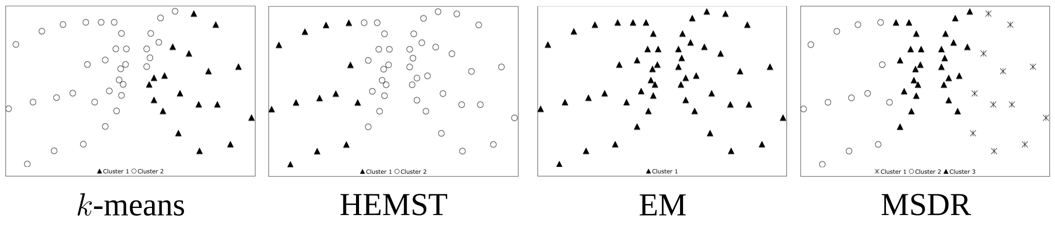


Fig. 6. Clusters with non-homogeneous densities.

Both HEMST and k-means are k-constrained. The EM algorithm determines the number of clusters through cross validation. Our MSDR algorithm selects the number of clusters with the largest second derivative. As we can see in Figure 4, k-means breaks both clusters and mixes them up into two undesired groups. HEMST and EM fragment one of the clusters, and only MSDR successfully identifies the clusters as desired. Similarly, in Figure 5, k-means fragments both the surrounding and the central cluster, while HEMST and EM manage to identify the central cluster, although the surrounding cluster is separated into two. Again, only MSDR successfully outputs the more appropriate cluster structures. In Figure 6, both k-means and HEMST tend to group points in a high density region into one cluster. EM only outputs one cluster, while MSDR outputs three clusters, grouping two high density regions into one cluster and identifying low density regions as two clusters on opposite sides. As can be observed, our MSDR algorithm is most successful in identifying the desired cluster structures. HEMST is slightly better than the k-means algorithm.

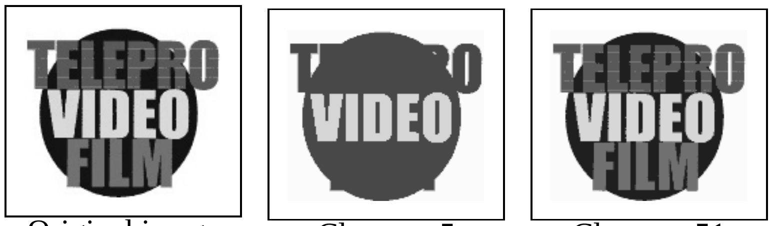
5.2 Image Color Clustering with EMSTs

We now compare our proposed EMST clustering algorithms to SEMST and ZEMST on images. All distinct colors in a given image are used to construct a Euclidean minimum spanning tree in the RGB color space. Each distinct color represents a node in the tree, and the edges are represented by the Euclidean distances between the RGB values of two nodes.

As mentioned in earlier sections, given k, SEMST produces clusters by removing the $k-1$ longest edges. ZEMST removes inconsistent edges without a preset k value. Typical issues each algorithm commonly faces include:

1. For a given k, simply removing the $k-1$ heaviest edges is not sufficient to obtain the desired cluster structure.
2. For an unknown k, removing inconsistent edges often produces an unnecessarily large number of clusters for a given input, or leads to undesired partitions. More problematically, the performance of the algorithm heavily relies on a set of constants that must be determined by the users.

These problems are illustrated in Figure 7. On the left is the original JPEG image containing 5328 different colors, even though to human eyes, there are only five different colors including the background color. The middle image is a result of color clustering using SEMST with $k = 5$, which removes the four heaviest edges to create five clusters. However, to human eyes there are only three colors left in the image. On the right, the color clusters are created by ZEMST. Even though the image looks very close to the original, the number of clusters is 51 which is far greater than the ideal five clusters.



Original image Clusters=5 Clusters=51

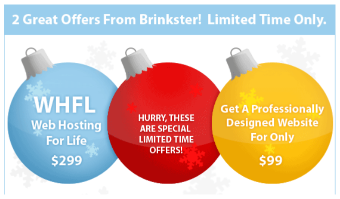
Fig. 7. Problems with color clustering.

Next we present the results of our second experiment, comparing our proposed algorithms to HEMST and ZEMST on a collection of 35 GIF and JPEG images, many of which contain a large number of colors.

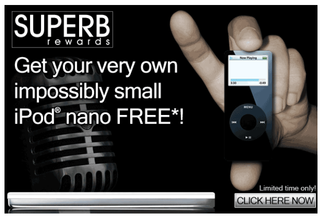
5.2.1 k-constrained Color Clustering

We now report the clustering results using k-constrained EMSTs, i.e. HEMST and SEMST, on three GIF images shown in Figure 8.

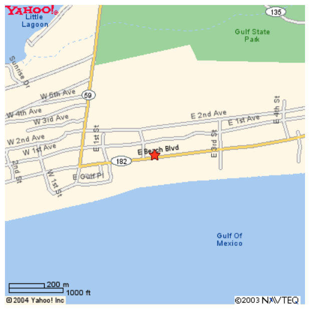
Figure 9 shows the results of our HEMST and the SEMST algorithm on the leftmost GIF image in Figure 8 containing 128 distinct colors. On the left are the color clusters produced by our HEMST algorithm, on the right are the clusters produced by SEMST. When $k = 2$ the output of our algorithm has managed to catch most of the objects in the original image, while the output of the SEMST algorithm was


2 Great Offers From Brinkster! Limited Time Only.
WHFL
Web Hosting
For Life
$299
HURRY, THESE ARE SPECIAL LIMITED TIME OFFERS!
Get A Professionally Designed Website For Only
$99

brinsker.gif



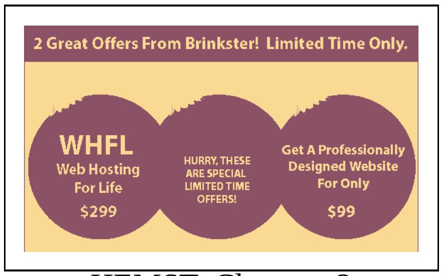
ipodnano.gif



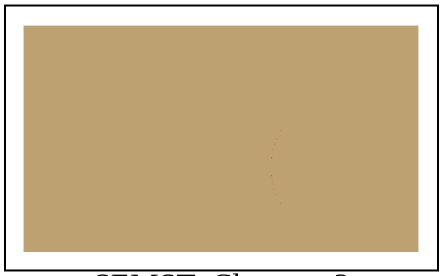
map.gif

Fig. 8. Image brinsker.gif, ipodnano.gif and map.gif before color clustering.

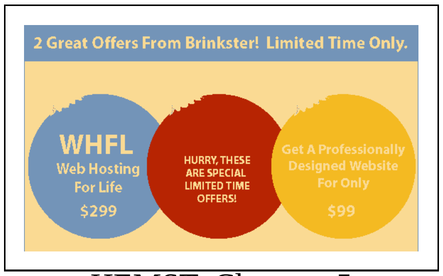
not able to detect any object in the image. When $k = 5$, our algorithm catches even more details of the original image.



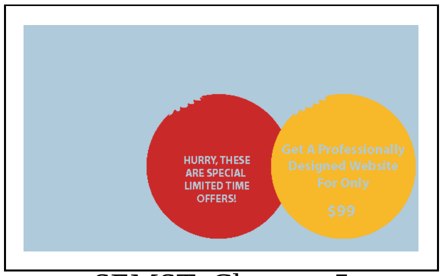
HEMST: Clusters=2



SEMST: Clusters=2



HEMST: Clusters=5



SEMST: Clusters=5

Fig. 9. Image brinsker.gif after color clustering using HEMST and SEMST.

Figures 10 and 11 show the clustering results of the two algorithms on the other two images in Figure 8. The iPod nano image contains 69 distinct colors. With our technique, when $k = 2$ it is sufficient to identify all the objects including the text in the image. The SEMST algorithm was not able to output anything significant. The map contains 189 distinct colors. With only two colors ($k = 2$), our algorithm allows us to see clearly all the street names and the location of the destination. The SEMST algorithm managed to output a red star which indicates the destination. When $k = 8$ (8 representative colors), the output of our algorithm is almost identical to the original one, while the SEMST algorithm still could not output the street names and other detailed information on the map.

We compared the two algorithms on 35 different images. Due to space limit, we do not show all the results. It is apparent that given a cluster number k, our HEMST algorithm is much more effective than the SEMST algorithm that simply removes

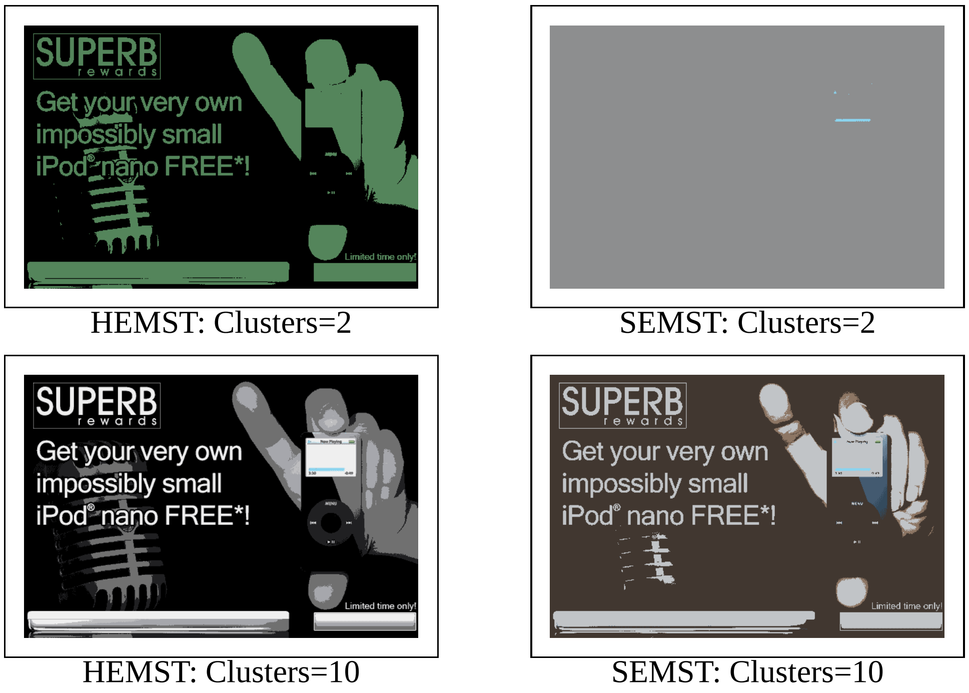


Fig. 10. Image ipodnano.gif after color clustering using HEMST and SEMST.

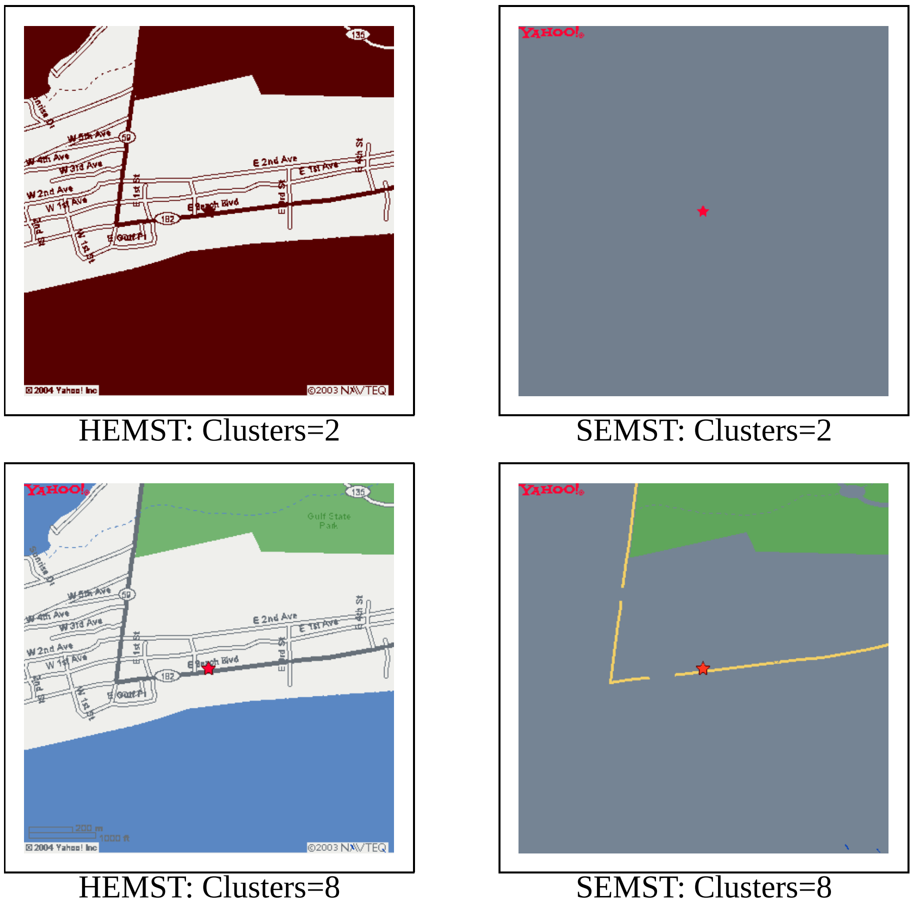
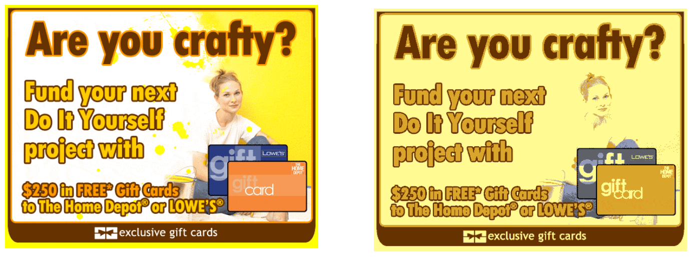


Fig. 11. Image map.gif after color clustering using HEMST and SEMST.

the $k-1$ longest edges from the tree. Our algorithm is able to pick a very few most representative colors from a large number of colors in the original image, and still manage to capture all the objects in the images.

5.2.2 Unconstrained Image Color Clustering

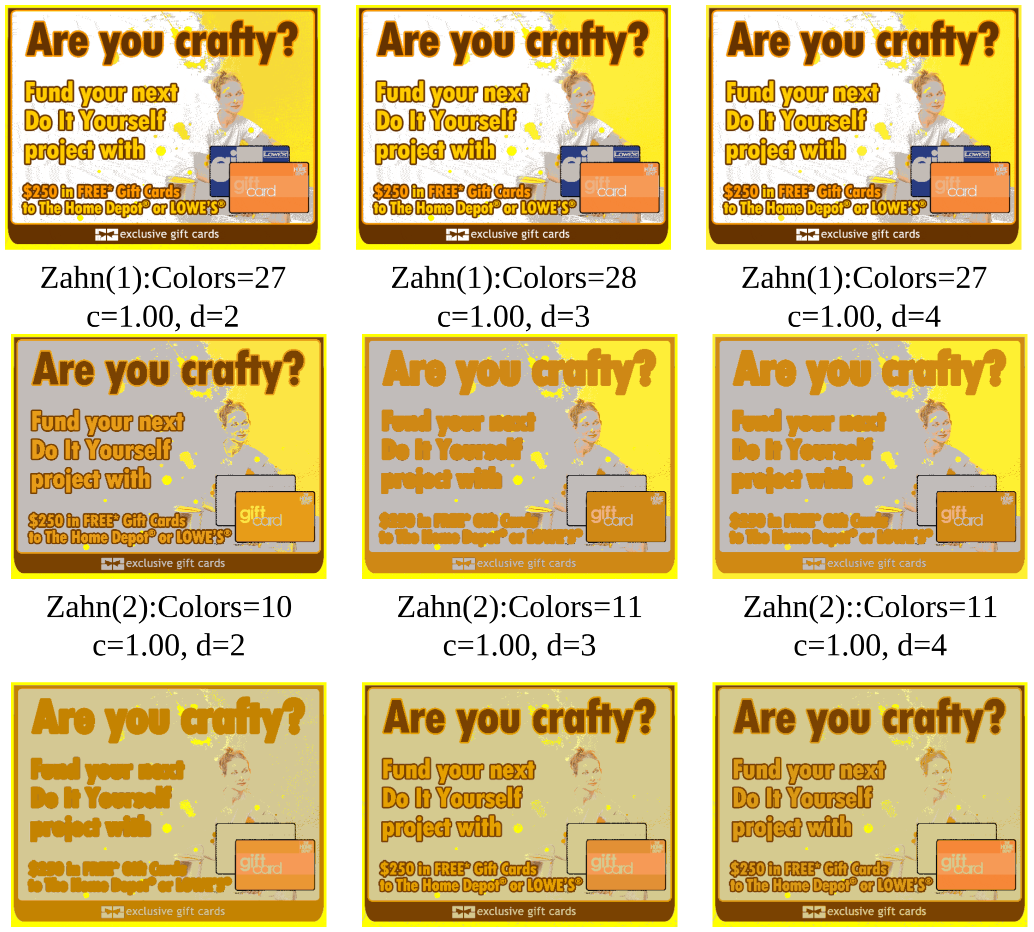
In this experiment, the desired number of clusters as well as the structure of color clusters are unknown to the algorithms. We compared our MSDR algorithm to Zahn's EMST clustering algorithm [16]. For Zahn's algorithm, we applied three different criteria to detect inconsistent edges in the tree as discussed in earlier sections: consider nearby edges on one side of the edge, consider nearby edges on both sides, consider the ratio of the average weights of nearby edges on both sides. Due to the complex nature of images, our MSDR algorithm selects the cluster number corresponding to a local minimum of the regression function in this experiment. For Zahn's method, we have to choose the size of the neighborhood explored, the number of standard deviations in excess to the average weight, often differently for different images in order to obtain better results. For each of the first two criteria, we choose three different neighborhood sizes: search depth into the tree $d = 2, 3, 4$, and three standard deviation factors $c = \{0.25, 0.75, 1.0\}$. We did not use the factor numbers suggested by Zahn because the algorithm could not output anything interesting with those factors. For the last criterion, we use the same search depths, and three different ratios $f = \{1.0, 1.25, 1.5\}$. Figure 12 shows the original image of a gift card and the result of our MSDR clustering algorithm. There are 58 colors in the original image. Our algorithm output eight color clusters and was able to capture nearly all the objects in the image.



Original Image MSDR: Colors=8

Fig. 12. The original image giftcard.gif and the output of our MSDR algorithm.

Due to space limit, Figure 13 only shows the results of the ZEMST algorithm with the number of clusters closest possible to our MSDR output using the three different criteria with different parameter choices. As can be observed, the quality of the output relies heavily on the parameter choices. In fact, the output cluster number varies from 2 to 38 as the choice of the parameters varies. In practice, determining the parameter combinations could be rather challenging. In our experiment, we tested all parameter combinations on all 35 images. The general observation is that our MSDR algorithm is able to produce a much smaller number of colors while preserving objects embedded in all images, compared to the ZEMST algorithm.


Are you crafty?
Fund your next Do It Yourself project with
$250 in FREE* Gift Cards to The Home Depot® or LOWE'S®
gift card
exclusive gift cards

Zahn(1):Colors=27 c=1.00, d=2 | Zahn(1):Colors=28 c=1.00, d=3 | Zahn(1):Colors=27 c=1.00, d=4

Zahn(2):Colors=10 c=1.00, d=2 | Zahn(2):Colors=11 c=1.00, d=3 | Zahn(2)::Colors=11 c=1.00, d=4

Zahn(3):Colors=13 f=1.25, d=2 | Zahn(3):Colors=12 f=1.25, d=3 | Zahn(3):Colors=12 f=1.25, d=4

Fig. 13. ZEMST using the 1st, 2nd, 3rd criteria with different parameter choices.

5.3 *Comparing MSDR, SFMST and k-means on UCI Datasets*

We compared our MSDR clustering algorithm with SFMST and k-means on four benchmark datasets from the UCI repository [11]. Table 1 gives the summary of the data sets.

Table 1
Benchmark Data Sets from the UCI Repository

Data Set	# of Instances	# of Attributes	# of Classes
Iris Plants	150	4	3
Pima-Indians Diabetes	768	8	2
Thyroid	215	5	3
Image Segmentation	2100	19	7

For the first three data sets: Iris Plants, Pima-Indians Diabetes and Thyroid, we compare the MSDR algorithm to several algorithms that have been reported by Päivinen [12], including the Scale-free MST (SFMST) algorithm, k-means with two different k values, and the standard MST algorithm. For Image Segmentation data (not reported by Päivinen), we compare MSDR to k-means with $k = 5$ and $k = 9$. Since all the data sets include class labels, we measure the purity (impurity) of each cluster using *entropy* $\sum_{i \in c} -p_i \log p_i$, in which c is the total number of classes of a data set, p_i is the probability that an instance belongs to class i in that cluster. We report the average entropy of clusters created by each algorithm. Lower entropy values imply purer clusters. To avoid bias towards a larger number of small clusters, we ignore the clusters containing less than 10 instances or 10% of the instances in the data set, whichever is smaller, when we compute the entropy.

Table 2 shows the results on the Iris Plants data. Our MSDR algorithm produced 3 non-trivial clusters. The first cluster contains the majority of *setosa* and only *setosa*, which conforms to the fact that *setosa* is linearly separable from the other two species. In fact, each algorithm manages to separate *setosa* from the other two, however, our MSDR algorithm is the only one that separates *versicolo* well from *virginica*. This also explains the very low (lowest) average entropy of the clusters produced by our MSDR algorithm.

Table 3 shows the results on the Thyroid data. Our MSDR algorithm produced 2 non-trivial clusters, of which the average entropy is significantly lower than that of the clusters produced by other MST based algorithms. The first cluster contains the majority of *Normal*. Clusters 3 to 6 solely contain the *Hypo* instances, while with the other two MST algorithms, *Hypo* instances can not be identified. k-means, on the other hand, separated instances of different classes quite successfully.

Table 4 shows the results on the Pima Indians Diabetes data. The instances in this data set are not well separated using the Euclidean distance measure. Every algorithm failed to successfully separate the instances according to their labels. The MSDR algorithm produced 4 non-trivial clusters, of which the average entropy is significantly lower than that of the clusters produced by all other algorithms.

The result of the Image Segmentation data is not reported by Päivinen. We instead compared MSDR to k-means with $k = 5$ and $k = 9$, which are chosen due to the large size of the data set. As shown in Table 5, our MSDR algorithm managed to separate all *sky* instances from the others, and successfully identified most of the *path* instances, which the k-means algorithm could not do with the same k values. The average entropy of the clusters produced by the MSDR algorithm is comparable to that of the clusters produced by the k-means when $k = 5$, but much better than when $k = 9$.

Table 2
Results on the Iris Data Set. In each section of the table, the first column is the clusters generated by a clustering algorithm. The last column is the entropy value of each cluster and the average entropy. The rest of the columns give the number of instances in each class.

	Seto	Vers	Virg	Entropy		Seto	Vers	Virg	Entropy
SFMST					MST				
C_1	44	1	0	0.15	C_1	1	45	30	1.06
C_2	1	35	28	1.09	C_2	36	0	0	0
C_3	0	0	17	0	C_3	0	4	7	0.95
					C_4	13	0	0	0
				Avg: 0.41					**Avg: 0.50**
k-means					k-means				
(k=5)					(k=3)				
C_1	0	19	2	0.45	C_1	33	0	0	0
C_2	0	2	27	0.36	C_2	0	46	50	0.70
C_3	22	0	0	0	C_3	17	4	0	0.999
C_4	0	29	21	0.98					
C_5	28	0	0	0					
				Avg: 0.36					**Avg: 0.57**
MSDR									
C_1	42	0	0	0					
C_2	0	40	0	0					
C_3	0	3	39	0.37					
				Avg: 0.12					

6 Conclusion

We have demonstrated that our proposed EMST-based clustering algorithms are very effective when applied to various clustering problems. Our HEMST clustering algorithm is k-constrained. The algorithm gradually finds a set of k representative points that serve as an “attractor” to points nearby, and outputs the inherent cluster structure subsequently. We have shown that our algorithm works much more reliably than the simple SEMST clustering algorithm. Our MSDR algorithm automatically determines the desired number of clusters. The objective function is defined to maximize the overall standard deviation reduction. Our algorithm does not require the users to select and try various parameter combinations in order to

Table 3
Results on the Thyroid Data Set.

	Nor	Hpe	Hpo	Entropy		Nor	Hpe	Hpo	Entropy
SFMST					MST				
C_1	37	1	2	0.45	C_1	118	8	30	0.98
C_2	96	20	0	0.66	C_2	7	24	0	0.77
C_3	0	0	12	0	C_3	18	3	0	0.59
B_n	17	5	16	1.43					
				Avg:0.64					**Avg:0.78**
k-means (k=5)					k-means (k=3)				
C_1	64	14	0	0.68	C_1	0	16	0	0
C_2	0	16	0	0	C_2	150	19	8	0.75
C_3	86	1	8	0.50	C_3	0	0	22	0
C_4	0	0	22	0					
				Avg:0.29					**Avg:0.25**
MSDR									
C_1	150	23	6	0.76					
C_2	0	0	13	0					
				Avg:0.38					

get the desired output.

The main challenge we have encountered when using the MSDR algorithm is runtime efficiency.We reduced the runtime of the MSDR algorithm significantly by storing in each node some information that enables the calculation of the standard deviation of each subtree in constant time. We have demonstrated that our MSDR algorithm outperforms other clustering algorithms, including SFMST, standard MST, and k-means on most of the benchmark data sets from the UCI repository.

We intend to further explore the potentials of MST based clustering algorithm in various data mining domains where cluster boundaries are inherently irregular. We will continue to study the rich properties of the MST clustering techniques and identify new challenges of applying those techniques in practice.

Table 4
Results on the Pima Indians Diabetes Data Set.

	Negative	Positive	Entropy		Negative	Positive	Entropy
SFMST				MST			
C_1	9	7	0.99	C_1	224	138	0.96
C_2	15	27	0.94	C_2	9	1	0.47
C_3	26	12	0.90	C_3	9	10	0.998
C_4	133	38	0.76				
C_5	32	37	0.996				
B_n	33	31	0.999				
			Avg:0.93				**Avg:0.81**
k-means				k-means			
(k=9)				(k=5)			
C_1	27	23	0.995	C_1	102	9	0.41
C_2	13	26	0.92	C_2	72	43	0.95
C_3	8	9	0.997	C_3	19	39	0.91
C_4	56	20	0.83	C_4	47	52	0.998
C_5	33	38	0.996	C_5	8	9	0.997
C_6	23	22	0.996				
C_7	80	7	0.40				
C_8	5	6	0.99				
			Avg:0.89				**Avg:0.85**
MSDR							
C_1	215	107	0.92				
C_2	16	17	0.999				
C_3	0	11	0				
C_4	9	2	0.68				
			Avg:0.65				

References

[1] T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the 4th Annual Symposium on Computational Geometry*, pages 252–257, 1988.

Table 5
Results on the Image Segmentation Data Set

	Brick	Cement	Foli	Grass	Path	Sky	Wind	Entropy
k-means								
(k=5)								
C_1	27	254	27	0	298	0	36	1.66
C_2	0	0	0	0	0	144	0	0
C_3	0	0	0	0	0	110	0	0
C_4	0	18	1	0	0	46	0	0.96
C_5	273	28	272	300	2	0	264	2.13
								Avg: 0.95
k-means								
(k=9)								
C_1	0	32	0	48	89	0	96	1.87
C_2	0	79	0	0	0	0	0	0
C_3	0	50	27	0	0	0	0	0.93
C_4	0	0	60	0	0	0	0	0
C_5	0	31	16	106	22	118	61	2.25
C_6	170	57	50	44	117	44	74	2.61
C_7	0	32	36	0	0	0	16	1.51
C_8	0	11	111	56	30	58	32	2.30
C_9	130	8	0	46	42	80	21	2.19
								Avg: 1.52
MSDR								
C_1	300	150	267	300	0	0	289	2.28
C_2	0	0	0	0	0	300	0	0
C_3	0	0	0	0	245	0	0	0
C_4	0	14	2	0	15	0	0	1.28
C_5	0	127	0	0	40	0	0	0.79
C_6	0	2	12	0	0	0	9	1.33
								Avg: 0.95

[2] D. Avis. Diameter partitioning. *Discrete and Computational Geometry*, 1:265–276, 1986.

[3] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *Proceedings of the 43rd FOCS*, pages 238–247, 2002.

[4] M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71:360–383, 2005.

[5] E. Demaine and N. Immorlica. Correlation clustering with partial information. In *Proceedings of the 6th RANDOM-APPROX*, pages 1–13, 2003.

[6] C. Eldershaw and M. Hegland. Cluster analysis using triangulation. In B. Noye, M. Teubner, and A. Gill, editors, *Computational Techniques and Applications: CTAC97*, pages 201–208. World Scientific, 1997.

[7] R. Gonzalez and P. Wintz. *Digital Image Processing (2nd Edition)*. Addison-Wesley, Reading, MA, USA, 1987.

[8] J. Gower and G. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 18:54–64, 1969.

[9] D. Johnson. The np-completeness column: An ongoing guide. *Journal of Algorithms*, 3:182–195, 1982.

[10] D. Lopresti and J. Zhou. Locating and recognizing text in www images. *Information Retrieval*, 2:177–206, 2000.

[11] C. J. Merz and P. M. Murphy. UCI repository of machine learning databases, 1998.

[12] N. Päivinen. Clustering with a minimum spanning tree of scale-free-like structure. *Pattern Recogn. Lett.*, 26(7):921–930, 2005.

[13] Y. Xu, V. Olman, and E. Uberbacher. A segmentation algorithm for noisy images: design and evaluation. *Pattern Recognition Letters*, 19:1213–1224, 1998.

[14] Y. Xu, V. Olman, and D. Xu. Minimum spanning trees for gene expression data clustering. *Genome Informatics*, 12:24–33, 2001.

[15] Y. Xu and E. Uberbacher. 2d image segmentation using minimum spanning trees. *Image and Vision Computing*, 15(1):47–57, 1997.

[16] C. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, C-20:68–86, 1971.