Exploring graph-based clustering and outlier detection algorithms

Li, J.

Chapter 2

A Scaled Minimum Spanning Tree Based Clustering Algorithm and Application on Image Segmentation

This chapter is based on the following publications:
Li, J. , Wang, X. , and Wang, X. . "A scaled-MST-based clustering algorithm and application on image segmentation." Journal of Intelligent Information Systems 54 (3) (2020): 501-525. DOI: 10.1007/s10844-019-00572-x

2.0.

Minimum spanning tree (MST)-based clustering is one of the most important clustering techniques in the field of data mining. Although traditional MST-based clustering algorithm has been researched for decades, it still has some limitations for data sets with different density distribution. After analyzing the advantages and disadvantages of the traditional MST-based clustering algorithm, this paper presents two new methods to improve the traditional clustering algorithm. There are two steps of our first method: compute a scaled-MST with scaled distance to find the longest edges between different density clusters and clustering based on the MST. To improve the performance, our second scaled-MST-clustering works by merging the MST construction and inconsistent edges' detection into one step. To verify the effectiveness and practicability of the proposed method, we apply our algorithm on image segmentation and integration. The encouraging performance demonstrates the superiority of the proposed method on both small data sets and high dimensional data sets.

Keywords: Minimum spanning tree, Clustering, Minimum spanning tree-based clustering, Image segmentation, Image integration

2.1 Introduction

Given N data points in R^d, where d denotes the dimension of data, a minimum spanning tree (MST) is a tree that connects all data points and the total weights of all edges is minimized and any data points can be reached from others by following the edges. In a complete graph with a large-scale data set of size N, there are $V = N$ vertices and $E = N(N-1)/2$ edges and cost of constructing a MST is $O(ElogV)$, roughly equal to $O(dN^2)$. As a widely used technique in many fields, MST has been researched for decades. It was first proposed by Otakar Boruvka in 1926 [17] and has been widely accepted in image segmentation [6, 161], classification [71], pattern recognition [176], clustering [162, 160] and machine learning.

As an important machine learning technique, clustering analysis is the process of dividing a set of objects into non-overlapping subsets, which aims to maximize the inter-cluster similarity and minimize the intra-cluster similarity. Due to its abundant applications in data mining and other fields, various techniques of clustering come into focus over the past decades, such as spectral clustering [23], hierarchical clustering [175], distance and density based clustering [55].

The cutting property of an MST is that edges with the smallest weight connecting any two subtrees of the point set must belong to the MST. Each line in the MST is the shortest distance between two subgraphs when the weight stands for the Euclidean distance between two end points [153]. Based on this, the basic idea of MST-based clustering is to find the inconsistent edges (the edges whose weight is significantly larger than the average weight of nearby edges in the tree) of the MST and then remove them. Since MST-based clustering was proposed by Zahn in 1971 [162], discussions regarding it occupies the dominant position in the related research. Compared to other clustering methods, MST-based clustering can separate the data sets with different shapes and sizes independent on the shape of clusters. Besides, the assumption that data points are grouped around centers or separated by regular geometric curve is not necessary. Traditional MST-based clustering algorithms usually use the Euclidean distance between two end points as the edge weight, and delete the longest edge to get two clusters. Recent researches on MST-based clustering focus more on the efficiency of the algorithm [69, 70, 152, 151]. Among the previous MST-based clustering techniques,

many successful ones benefit from improving the quality of clustering. MST-inspired clustering can apply to the outlier detection technique [154]. At the same time, the quality of MST-based clustering algorithm is often improved by removing outliers [153]. Besides, neighborhood density estimation is a crucial technique in the MST-based clustering [102]. These investigations, however, disregards the data sets with different density, that means they can't well recognize the different density clusters. A new approach is therefore needed for the construction of MST and MST-based clustering for this kind of data distribution.

Image segmentation is the process of partitioning an image into multiple disjoint subsets such that each subset is more meaningful and easier to analyze. In computer vision, image segmentation is one of most fundamental problems. Because of its importance in computer vision, many different segmentation schemes have been proposed over the last decades. Among them, MST-based segmentation is substantially related to the figure-based clustering [41]. The clustering or grouping of image pixels are performed on MST. The connection of graph vertices satisfies the minimal sum on the defined edge weights, and the partition of a graph is achieved by removing edges to form different subgraphs.

In order to overcome the limitations of traditional MST-based clustering in some special data sets, we present a new method, called scaled-MST, which can apply to finding the clusters with different density and irregular boundaries. Application of edge scaling makes positive contribution to our algorithm. Furthermore, according to the graph theoretical methods for image processing, we apply our MST-based clustering method to the image segmentation and integration. Our contributions mainly focused on four aspects. Firstly, we propose a new scaled-MST algorithm, which works by using scaled edge weights to replace the Euclidean distance between two end points. Secondly, we improve the traditional MST-based clustering by deleting the points which have been clustered such that the data space could be reduced largely. Thirdly, we test our method on high dimension (10000) image data sets and then apply to image segmentation. Finally, we integrate five pictures of the image data and improve the efficiency of high dimensional data sets.

The rest of paper is organized as follows: In Section 2, some existing work on MST-based clustering and image segmentation techniques is reviewed. Then we present our proposed scaled-MST in Section 3. Section 4 highlights the perfor-

mance evaluation of different clustering methods. We compare our method with k-means, spectral clustering, FMST [174], cciMST and PSR-MST. Finally, we make a conclusion and future work in Section 5.

2.2 Related work

There are two main categories related to our proposed method: MST-based clustering algorithms and image segmentation and integration.

2.2.1 MST-based clustering algorithms

The first step in modern MST-based clustering algorithm is to construct a MST for N data points in Euclidean space. Every pair of data is connected by a line, so there are $N(N-1)/2$ lines in a complete graph (an undirected graph in which every pair of distinct vertices is connected by a unique edge), however, only N-1 lines are remained in the MST. The cost of traditional MST algorithm is approximately $O(N^2)$ [31]. This time factor limits the application of MST-based clustering methods to massive data sets. Several studies have been done to this problem, which lead to better MST construction efficiency.

According to the cutting property of MST, the next step of MST-based clustering is to find the inconsistent edges to divide the tree into clusters. It was first proposed by Zahn in 1971 [162]. The inherent separation was used in this paper to emphasize that the separation relies solely on inter-point distances within the data sets. The inherent relationship between MST and clusters makes the image segmentation performed in an implicit way. However, it is weak in processing the clusters with complex structure.

To reduce the dependence of clustering performance on parameters, He et al. proposed a threshold criterion for the single linkage cluster analysis in 2004 [59]. They advise an approach to auto-detect the thresholds to avoid the human input of parameters. This method has a good performance on spatial data and arbitrary shaped clusters. However, it is not good enough for high dimensional data sets.

In the real-word data, there are many outliers that usually have negative influences on the clustering results. To address this issue, Wang et al. proposed an efficient MST-inspired clustering algorithm. It works by computing local density

factors for each data point to find the density-based outliers during the construction of an MST [153]. The application of iDistance significantly reduces the computation time of LOF (Local Outlier Factor). Although this method is particularly suitable for high dimensional data sets, it does not work well for different density data sets.

In order to develop high efficiency algorithms for MST-based clustering, increasing researchers pay close attention to the efficiency of the construction of MST which is fit for different data sets. Many researchers focus more on the efficiency of the construction of MST and have found fast and good algorithms to fit different data sets. So far, parallel algorithm is the best method to improve the efficiency among all efficient algorithms [29]. However, it is a big challenge for the study of the parallel random access machine (PRAM) model of computation.

By using a centroid based nearest neighbor rule, a MST-based clustering algorithm is introduced with generating a sparse Local Neighborhood Graph (LNG) and then constructing the approximate MST from LNG in [69]. Experimental results reveal that the computational time has been reduced significantly by maintaining the quality of the clusters obtained from the MST.

Divide-and-conquer scheme can be used to produce an approximate MST [174]. There are two stages in this algorithm, divide-and-conquer stage and refinement stage. It is efficiency comparing with other MST algorithms. And the experiment results demonstrate the outstanding performance of this method on clustering.

Relying on nothing more than a compressed quadtree data structure to compute approximate EMST(Euclidean Minimum Spanning Tree), the algorithm in [9] eliminates the exponential ϵ dependencies to reduce the time consumption. This algorithm achieves its efficiency by being sloppier in approximating bichromatic closest pairs between well-separated pairs in situations where there is significant EMST weight in the vicinity of the well-separated pair could be inferred.

R.Jothi et al. proposed two efficient algorithms namely partition based near neighbor graph using Bi-means partitioning (BNNG) and partition based near neighbor graph using k-means partitioning (KNNG) to obtain MST in the context of clustering in a less than quadratic time [70]. A novel centroid based nearest neighbor rule is presented in this paper and the experiment results reveal the good performance of the algorithm.

By improving the Density Peaks Clustering (DPC) algorithm, Lv et al. proposed a novel MST-based clustering method through the cluster center initialization algorithm [103]. Geodesic distance is employed to find the inconsistent edges. They can get a better performance on image data sets.

Many MST-based clustering algorithms are not practical for large scale high dimensional data sets. To alleviate this defect, Wang et al. proposed a fast two-level approximate EMST algorithm for high dimensional data [152]. Their method identify the boundary points by running outlier detection algorithm, which reduce the data set to certain extent and improve the efficiency. Therefore, the efficiency can be improved. Besides, it conducted a KNN search to complete an AMST construction process. However, the dimension of data set in the experiments is not large enough.

2.2.2 Image segmentation and integration

Image segmentation is a fundamental problem in computer vision, which has been researched for many years. There are many ways for image segmentation. These methods are categorized into five classes under a uniform notation: the minimal spanning tree based methods, graph cut based methods, the shortest path based methods and the other methods that do not belong to any of these classes [118]. A number of researches have been focused on these methods. Among them, the minimal spanning tree based segmentation is substantially related to the graph based clustering [41]. The clustering or grouping of image pixels are performed on the minimal spanning tree. The connection of graph vertices satisfies the minimal sum on the defined edge weights, and the partition of a graph is achieved by removing edges to form different subgraphs.

Zhang et al. make a survey of unsupervised methods in the field of image segmentation [165]. In this paper, they examine the unsupervised objective evaluation methods that have been reported in the literature. An extensive evaluation of these methods is presented. The advantages and shortcomings of the underlying design mechanisms in these methods are discussed and analyzed through analytical evaluation and empirical evaluation.

In 2012, Vella et al. proposed a new image segmentation approach through a hierarchy of minimum spanning trees [148]. By using graph theory, it can get a good result for image segmentation.

Using clustering to process image has drawn much attention for the last decades. Dhanachandra et al. proposed an image segmentation method using k-means clustering algorithm and subtractive clustering algorithm [36]. They use subtractive cluster to generate the initial centers which can be used in k-means algorithm. The proposed method gains an excellent performance when process the initial image and the segmented image by partial contrast stretching and median filter separately. For social image understanding, Li et al. proposed an image retrieval method using weakly supervised deep matrix factorization in 2017 [92].

By developing a new predicate for the cutting criterion, Saglam et al. proposed a Prim sequential representation of MST (PSR-MST) and applied on image segmentation [134]. A threshold value c and a parameter l are tuned to make a better performance. The result is competitive with some other image segmentation algorithms. However, the unfit values of the parameters may cause incorrect segmentation results.

To fit the highly sparse high dimensional data sets of image, Wang et al. proposed an efficient approximate EMST algorithm in 2018 [151]. Using Euclidean distance and the Cosine similarity to compute the inner product, the construction of MST is more efficient. Unfortunately, the work pays a little attention to the curse of dimensionality.

2.3 The proposed method

According to the cutting property mentioned above, MST-based clustering begins by constructing an MST of given data sets and then proceeding to remove the inconsistent edges(e.g. longest edges) to generate clusters. Based on this, a prominent shortcoming of traditional MST-based clustering algorithm is that the cut on every edge can result in two subtrees, which may lead to a partition without sufficient evidence. Figure 2.1 shows two examples about Prim's algorithm on different density data sets. The following step of traditional MST-based clustering algorithm is to remove the longest edge, but the result exhibits that the different density clusters cannot be separated well. It is obvious that the different density clusters cannot be separated well.

Comparing with traditional MST algorithm, our scaled-MST can find the edge connecting two different density groups by using scaled edge weight. Cutting

of this edge can generate two different density clusters. Therefore, for the different density data sets, the quality of clustering can be improved to some extent.

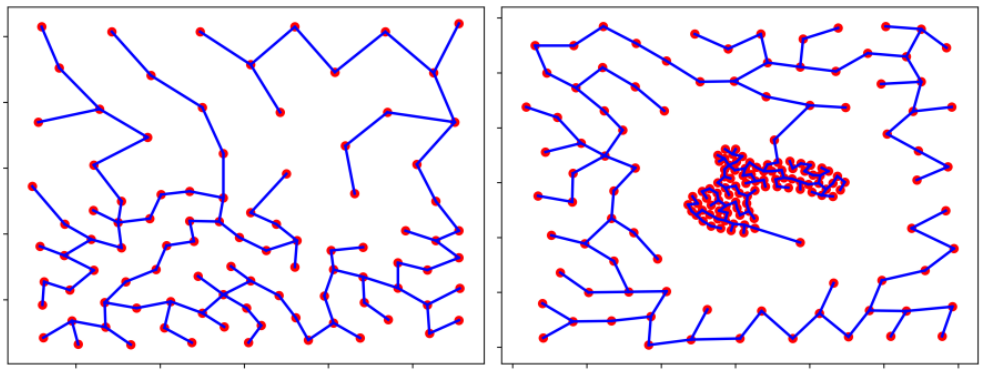


Figure 2.1: The result of Prim on different density data sets.

2.3.1 Scaled-MST

Given data sets $R = \{x_1, x_2, x_3, ..., x_n\}$ of size N in d-dimensional Euclidean space, the Euclidean distance between two data points can be given as:

$$eu\_dist(x_1, x_2) = \sqrt{\sum_{i=1}^{d} (x_{1i} - x_{2i})^2} \tag{2.1}$$

In traditional MST algorithm, the edges are weighted using this kind of distance measure. We use ω_e and ω_s to denote the Euclidean distance and the scaled distance of edge connected two end points respectively. Our scaled-MST algorithm modifies the Prim's algorithm by using the scaled distance:

$$scaled\_dist(x_1, x_2) = \frac{eu\_dist(x_1, x_2)}{scaled\_threshold} \tag{2.2}$$

The *scaled_threshold* is a value scaled the Euclidean distance. In our algorithm, there are three methods to compute the *scaled_threshold*:

1. a changeless value, such as 2;
2. the weight (Euclidean distance) of the edge last added to the tree;
3. the standard deviation added mean value of the "*Sliding Window*".

This algorithm is easy to implement. To be as general as possible, we use N to denote the size of data set and two arrays: $node\_finished$ and $node\_unfinished$ to judge whether the node has been added to the tree. We use an array to store the MST, called $edge\_arr$. Every element of the array is one edge, which contains $start\_point$, end_point and $ratio$ (the scaled distance between $start\_point$ and end_point). Besides, two arrays are employed in our algorithm: $index\_arr$ and $ratio\_arr$. The length of these two arrays is equal to N. The indexes of these two arrays are corresponding to the indices of data set. In the beginning, all data points except start point are in the $node\_unfinished$, and $node\_finished$ including the start point. With the construction of MST going on, the point is added to the tree one by one and we move the point from $node\_unfinished$ to $node\_finished$. This means, every time we move one point, one edge should be added to the $edge\_arr$. The method to determine which point should be moved is to select the point with the smallest ratio in $ratio\_arr$. After we compute ω_e of the edge, we compute ω_s according to Eq.2.2. The details of our scaled-MST algorithm are given in Algorithm 2.

Furthermore, in order to make the calculation faster, we employ a new method to read the data and compute the distance. Supposing that the data is stored in the file by sparse matrix, we just read the index and the value without considering 0 value. We compute the edge weight just by non-zero value of two end points.

There are some special situations in calculating the distances. For example, when two points are very close to each other, ω_e may be very small, even approach zero. According to Eq. 2.2, ω_e may be the denominator in the second method of calculation of ω_s. Therefore, we regard these two points as the same one, add them to the $node\_finished$ directly without calculating of ω_e and ω_s. Without loss of generality, our method has no specific requirements on the dimension of data sets. To understand our algorithm and compare with the traditional algorithm, Prim and our scaled-MST algorithm are presented in Algorithm 1 and Algorithm 2 respectively.

2.3.2 Scaled-MST-based clustering

In the clustering phase, a label array ***Label*** is utilized to identify the different clusters. The data points in the same group have the same label. A variable $label$ is employed to denote the number of clusters. The initial value of $label$ is -1 and

Algorithm 1 Basic Prim MST algorithm.

Require: a set of N data points
Ensure: MST

1: Initialize any point as start_point p, set its parent to be -1, its distance to be 0, mark it finished, all other points unfinished, set the parents of points unfinished to be index of p
2: Calculate the distances between p and unfinished points, store them into distance array
3: Move the point with the smallest distance from unfinished to finished, say q
4: Add the edge $(p, q, eu\_dist(p, q))$ to MST
5: Compute the distances between q and all other unfinished points
6: **if** the distance is smaller than current value in distance array **then**
7: Update the distance array with new distance
8: Update the parent array with current_point
9: **end if**
10: Repeat steps 3 to 8 until all points are finished
11: **return** MST

it increases with the proceeding of cutting the longest edge.

From Algorithm 2, we can get a scaled-MST with parent array and ratio array. Inspired by the traditional MST-based clustering algorithm, the following step of our scaled-MST-based clustering algorithm is to cut the longest edge to generate clusters. However, the cutting of longest edge in MST-based clustering may produce small sets of isolated nodes. To alleviate this defect, we define *min_point* and *max_point* for cluster:

$$min\_point = ROUND\left(\sqrt{\frac{N}{d}}\right) \tag{2.3}$$

where $ROUND$ is a function that round down a number to the nearest integer. N denotes the size of data set, and d denotes the dimension of data set.

$$max\_point = N - min\_point \tag{2.4}$$

The data number in an effective cluster should be between the *min_point* and *max_point*. Therefore, an array *copy_label* is used to record the labels before cutting the edge. If the cutting of an edge results in unfit clusters, we cancel cutting the edge and continue to cut the next longer edge. Besides, a variable

Algorithm 2 Scaled-MST algorithm

Require: a set of N data points
Ensure: an MST represented by *edge_arr*, a parent array and a ratio array
Let ***edge_arr*** denote the generated MST
Let ***ratio_arr*** denote N ratios
Let ***index_arr*** denote N data indices
Let ***eu_dist***(p, q) return Euclidean distance between point p and point q
Choose the first point read from the file as the start point, s
Initialize node_finished with s, and node_unfinished with all other points
for $i = 1$ to N **do**
    index_arr$[i]$ = index of s
    ratio_arr$[i]$ = ***scaled_dist***$(s, i_{\text{th}}$ point in data set$)$
end for
Choose another point p from data points such that p is nearest to s
Add the edge denoted by s, p, ***ratio_arr***$[s]$ to ***edge_arr***
Move p from node_unfinished to node_finished
while node_unfinished is not empty **do**
    for q in node_unfinished **do**
        temp_dist = scaled_dist(p, q)
        if temp_dist $<$ ***ratio_arr***$[q]$ **then**
            Update ***ratio_arr*** with the scaled_dist
            Update ***index_arr*** with the index of p
        end if
        Choose point r with smallest ratio in ***ratio_arr***
        Add the edge denoted by p, r, ***ratio_arr***$[p]$ to ***edge_arr***
        Move r from node_unfinished to node_finished
        Update p with r
    end for
end while
return *edge_arr*

cut_threshold is employed to determine the termination of clustering. If ω_s of current cutting edge is less than *cut_threshold*, the clustering process would stop. Our scaled-MST-based clustering can be summarized in Algorithm 3.

Algorithm 3 Scaled-MST-based clustering algorithm.

Require: an MST and corresponding ratio array
Ensure: a label array
1: Let $\boldsymbol{N}$ denotes the size of the ratio array
2: Let ***Label*** denotes the cluster labels with size $\boldsymbol{N}$
3: Initialize ***Label*** with *label* -1
4: Compute the standard deviation and mean value of the ratio array as *cut_threshold*
5: Compute the min_point and max_point values according to Eq. 2.3 and Eq. 2.4
6: Sort the ratio array in non-increasing order
7: Choose the longest edge in MST as current_edge according to the ratio array
8: **while** the weight of current_edge > *cut_threshold* **do**
9: Cut current_edge to get two partitions, $p1$, $p2$, the size of these two partitions are s_1, s_2
10: **if** min_point $< s_1, s_2 <$ max_point **then**
11: Delete current_edge from MST
12: *label* += 1
13: Label the points in $p1$ with *label* and points in $p2$ with *label*+1 in ***Label***
14: **end if**
15: Select the next longer edge as current_edge
16: **end while**
17: **return** ***Label***

2.3.3 Our second scaled-MST-based clustering algorithm

In our first scaled-MST-based clustering algorithm, we start from arbitrary point in data sets. To gain better performance, we start from one point in the densest cluster (the cluster containing maximum points in given radius area). Therefore, we add a new parameter, *start_point*, as the input of our scaled-MST. In order to find the point in the densest cluster, we need do some preprocessing. We use Prim to compute an MST and sort the edges in non-decreasing order. The start point of the edge with the smallest distance in Prim's MST is the point in the densest cluster.

MST-based clustering algorithms usually need to construct an MST at first and the next step is to cluster the data set, which may increase the cost of the time. In order to improve the efficiency of clustering, our method merges the construction of MST and MST-based clustering into one step. When we complete the construction of partial MST, the points in the partial MST can be regarded as one cluster. To start a new cluster means to start computing a new MST with remained points. For our purpose, we delete the points after they have been clustered such that the data sets can be reduced in the following computation. To implement, we modify our scaled-MST algorithm by adding an exit condition, *new_cls_th*, to judge whether we should start a new cluster.

From the second method of computing the *scaled_threshold* (Eq.2.2), we know that the *scaled_dist* of points in the same cluster is close to 1, for that the corresponding *eu_dist* is close to each other. However, the *scaled_dist* between two points in different clusters increases suddenly. Inspired by this, our second scaled-MST-based clustering algorithm computes the standard deviation and mean value of the edges added to MST as *new_cls_th*. An array *temp_tree* is used to record the edges of current MST. We use Algorithm 2 to add edge to *temp_tree*. Then we can get the mean value of *temp_tree* :

$$\bar{\omega} = \frac{\sum_{i=1}^{N} \omega_{si}^2}{N} \tag{2.5}$$

where N denotes the size of *temp_tree*, ω_{si} is the edge weight (represented by scaled distance). Then we can compute *new_cls_th* to judge whether to exit current construction of *temp_tree*:

$$new\_cls\_th = \bar{\omega} + \sqrt{\sum_{i=1}^{N} (\omega_i - \bar{\omega})^2} \tag{2.6}$$

To make our method more scalable, we employ a structure, called "*Sliding Window*", to identify whether a point is noise. The "*Sliding Window*" stores the edge weight of given number edge. The given number is called "window size", which is given randomly. In general, we set this value 5. Then we compute the standard deviation and the mean value of the window, comparing with a given *noise_threshold* to identify noise. The *noise_threshold* is given according to

experience. This can be used to find the noise and delete it.

As mentioned above, *node_finished* remembers all points added to MST, including the points which have been labeled. Therefore, after a cluster was found, we label the points in this cluster with an auto-increasing value *current_label*. To improve the readability, the details of our second MST-based clustering algorithm are shown in Algorithm 4.

Algorithm 4 Our second scaled-MST-based clustering algorithm.

Require: a set of N data points, *noise_th*
Ensure: a label array
Let ***SW*** denote the "Sliding Window"
Let ***Labels*** denote the label of data points
Let ***node_finished*** denote the points added to MST
Let ***node_unfinished*** denote the points not added to MST
Let ***ratio_arr*** denote *scaled_dist* of pair points
Initialize the size of ***SW*** with 5, ***Labels*** with -1, ***node_unfinished*** with N data points
Use Prim to get an MST
Sort the MST in non-decreasing order according to the edge weight
Initialize *current_label* with 1
for ***current_edge*** in MST **do**
    Choose the ***current_edge*** and next 4 edges to add to the ***SW***
    Compute the standard deviation and mean value of ***SW*** as *stdm*
    if *stdm* > *noise_th* **then**
        Label the end points of the edges in ***SW*** as noise
    else
        Choose the *start_point* of ***current_edge*** as s
        Use scaled-MST algorithm to obtain a *temp_tree* with exit condition Eq.(2.6)
        for all point in *temp_tree* **do**
            if ***Labels***[point] = -1 **then**
                Labels[point] = *current_label*
            end if
        end for
        current_label += 1
    end if
end for
return ***Label***

2.3.4 Image segmentation and integration

Our scaled-MST-based clustering algorithm can apply to image segmentation by grouping the image pixels. The main idea is to extract the useful features of the picture and translate the color pixels into high dimensional data sets. Then we use scaled-MST algorithm to find the inconsistent edges and remove them to get sub-graphs. We use different colors to show different clusters in one picture. After clustering, we can get several groups in one picture such that the different objects can be easily identified.

In this section, we propose a new method of image integration. The key point of the image integration is to determine whether to merge clusters in different pictures. Two conditions are necessary to the integration of two clusters. An important merge condition is that the number of similar points in two clusters should reach a certain range. To determine the similar points, we use Support Vector Machine (SVM) linear classifier to predict the labels of data points. SVM is a kind of supervised learning model which is often used in classification or categorization.

Generally, we divide data sets into train data (one picture) and test data (another picture). We treat the cluster labels we generate in image segmentation phase as train labels. By training the image data and labels in one picture, we can get a model to predict the labels of image data in another picture. Then we compare the predicted labels in one cluster with the labels we get in image segmentation phase. If more than half of the cluster labels are the same, we determine another condition. If not, the two clusters are not necessary to be merged. Besides, we can use KNN classifier to predict the labels.

Before we compute another merge condition, we define the mean value of a cluster:

$$\bar{\omega} = \frac{\sum_{i=1}^{N} \omega_i^2}{N} \tag{2.7}$$

where N is the number of data points included in the cluster, ω_i is the Euclidean distance between point i and its nearest neighbor. Another essential merge condition is:

$$inter\_dist < 2 \times (\bar{\omega_1} + \bar{\omega_2}) \tag{2.8}$$

where $inter\_dist$ denotes the minimum Euclidean distance between two clus-

ters.

To improve the efficiency, we employ cover tree [15] structure to find the nearest neighbor of points in data sets. Cover tree can quickly find the nearest neighbors of points, especially the high dimensional data points. In our algorithm, we construct a cover tree for all data points in one cluster when computing the second merge condition.

Our method can apply to the new object recognition. After using our scaled-MST-based clustering method, we can easily get several clusters, then we compute the clusters in objective picture, using the two conditions to determine whether to merge. For our purpose, we would be more interested in those clusters whose intra-cluster distance is much smaller than the inter-cluster distance. Therefore, we calculate all intra-cluster distance and inter-cluster distance. If the two clusters dissatisfy the two conditions, it is believed that the cluster is a new object to a large extent.

2.3.5 Time complexity analysis

As discussed earlier, our first scaled-MST-based clustering consists of two steps, construction of scaled-MST and MST-based clustering. Similar to traditional MST-based clustering algorithm, the time complexity most depend on the computation of scaled-MST, $O(N^2)$. For our second scaled-MST-based clustering, we employ Prim and sort edges. The total time cost consists of three steps: $O(N^2)+O(NlogN)+O(N^2) = O(N^2+NlogN)$. In the image integration phase, we separate a large picture into several small pictures. Supposing that we divide the large picture (containing N data points) into m pictures, each picture contains M data points, $m = N/M$. The time complexity of our method on this large picture can be given as follows:

$$O(\frac{M(M-1)}{2}m) = O(\frac{\frac{N}{m}(\frac{N}{m}-1)}{2}m) = O(\frac{N}{2}(\frac{N}{m}-1)). \tag{2.9}$$

The complexity of traditional method is $O(\frac{N(N-1)}{2}) = O(N^2)$. Therefore, if we set m to be a large value, the time consumption can be reduced to some extent.

2.4 Evaluation

In this section, several sets of experiments are conducted to evaluate clustering algorithms. We compare the effects of the proposed method with five other methods including k-means, spectral clustering, FMST [174], cciMST [103] and PSR-MST [134]. k-means is a classical clustering method which was first used in 1967. The results of k-means are greatly influenced by the choice of the number of clusters and the initial centroids. Besides, k-means has very excellent performance in separating the spherical clusters. Spectral clustering is a widely used clustering method. Comparing with k-means, Spectral clustering shows greater adaptability and superiority among different distributions of data sets. Zhong-MST uses divide-and-conquer scheme to produce MST and has highly efficient on large data sets. CciMST tries to find inconsistent edges through cluster centers. The application on image data shows a good performance of cciMST. PSR-MST is a popular image segmentation method, which can identify the transition edges in MST. We evaluate these methods on synthetic data sets, real data sets and image data sets of Berkeley Segmentation data sets.

There is no specific input parameter for our first scaled-MST-based clustering method. As for our second scaled-MST-based clustering, the input parameters can be summarized as the size of "*Sliding Window*", *noise_th* and exit condition. In our algorithm, three methods are used to compute *scaled_threshold*. We will specify the value of parameters and choose different methods for different data sets as needed in the subsection.

We implement all our algorithms in Python. All the experiments were performed on a computer with Intel® Core™ 3.2GHz i5-3470 CPU and 4 GB RAM. The operating system is Windows 7. The source codes for FMST and cciMST are from the original author and the source codes for k-means and spectral clustering are from the Python package. We implement PSR-MST in Python according to the pseudocode in [134].

2.4.1 Results on synthetic data sets

To show the excellent performance of our proposed method, we first test two dimensional data sets. The results of FMST and scaled-MST are shown in Figure 2.2. It is obvious that the longest edge of FMST is in the sparse area.

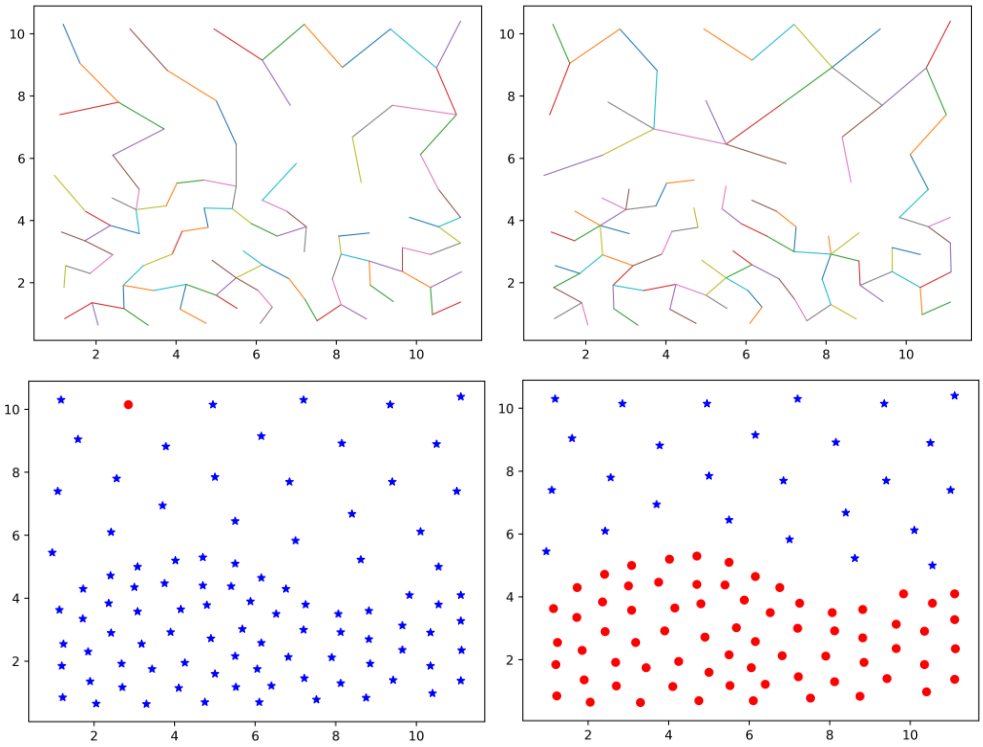
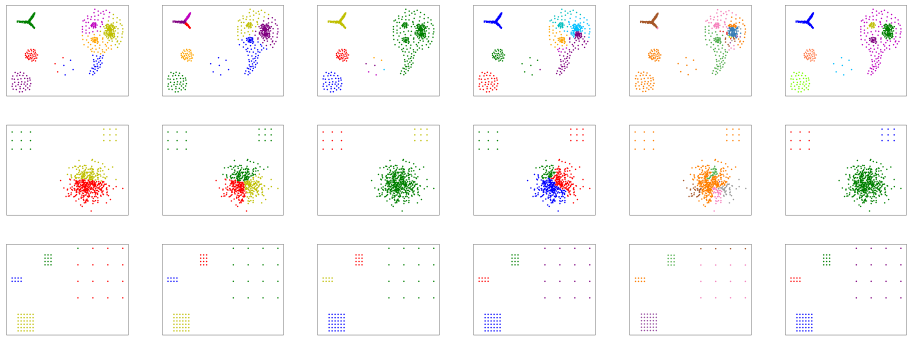


Figure 2.2: Upper left: FMST result, upper right: our scaled-MST result, lower left: clustering result of FMST, lower right: clustering result of scaled-MST.

That is, two end points of the longest edge is in the same cluster. As discussed earlier, the cutting of the longest edge in FMST results in an island. However, the longest edge of scaled-MST is the only edge connecting the different density clusters. So it can separate two different density clusters well. The red circles in Figure 2.2 denote the first cluster and the blue stars denote the second cluster.

For two dimensional data sets, we evaluate the results of our second scaled-MST-based clustering method by observing the different point colors of different clusters. Three two dimensional synthetic data sets with different densities are employed to test our clustering method. On the first row of Figure 2.3, it is clear that only our scaled-MST algorithm can separate the data sets successfully. There are total 8 clusters in this data sets. If we choose 8 as the cluster number, FMST could not separate the green points into 4 clusters and instead treat the noise as a cluster. Obviously, k-means, spectral and cciMST clustering are slightly weak



(a) k-means (b) spectral clustering (c) zhong-MST (d) cciMST (e) psr-MST (f) **scaled-MST**

Figure 2.3: Clustering results of (a)k-means, (b)spectral clustering, (c)FMST, (d)cciMST, (e)PSR-MST, (f)**scaled-MST**.

on clusters inside cluster. As for PSR-MST, if we set the parameter l to be 80, we can get a better result for this data set. We choose the third method we discussed above to get the *scaled_threshold*. The window size is set to be 8. The exit condition we set for this data set is: $new\_cls\_th = mean(sw) + 2 * (std(sw))$, where sw denotes the "*Sliding Window*" and std is used to compute the standard deviation. The $noise\_th$ is set to be 10.

On the second row of Figure 2.3, k-means separates the largest cluster into two clusters. Spectral and cciMST clustering separate the largest cluster into 3 clusters. We set l to be 200 for PSR-MST to get a better result. FMST and our method can successfully separate the data sets into 3 clusters and identify the noise. We modify the third method of computing *scaled_threshold* by only using the mean value without standard deviation. The window size is set to be 5. The exit condition is computed by Eq.(2.6). The $noise\_th$ is set to be 5.5. On the last row of Figure 2.3, k-means identifies a red point to be green incorrectly. Besides, all the other methods do well on this data set. The parameter values in our method are the same as the values of the first row data sets.

Furthermore, we consider CHAMELEON data sets from [74]. The CHAMELEON data sets consist of clusters of different size, shape and orientation. Due to that the performance of k-means is greatly influenced by the input

parameter (the number of clusters) and the selecting of initial centroids, the result of k-means is not always satisfactory. An inappropriate choice of k may yield poor results. Although k-means has an outstanding performance on spherical clusters, it cannot separate the arbitrary shapes smoothly. The results of spectral clustering on CHAMELEON data sets are not good enough. For the first line, FMST and our method can separate the six clusters well. cciMST merges the two blue clusters into one cluster and separates the orange clusters into two clusters. We set the parameter l in PSR-MST to be 180 to get this better result. k-means, cciMST and our method all behave wonderfully on the second data set with 6 clusters. Spectral and FMST identify the noises to be a cluster and merge different clusters to be one. For this data set, we set l of PSR-MST to be 60 to get this better result. Obviously, only our method can separate the 9 clusters correctly in the third and the fourth line. The parameter l is set to be 300 and 660 separately for PSR-MST. In the case of CHAMELEON data sets, we choose the second method to compute *scaled_threshold*. The window size is set to be 5. The exit condition is computed according to Eq.(2.6). The *noise_th* is set to be 8.

Compared with the other five algorithms we mentioned above, the results from this set of experiments show that our scaled-MST-based clustering is more suitable for data sets with different density clusters (Figure 2.4).

2.4.2 Results on real data sets

In this set of experiments, our first scaled-MST based clustering method is tested on low dimensional data sets. The parameters of the largest and least points are computed by Eq. 2.3 and Eq. 2.4. The scaled-MST start to construct MST from the first point we read from the file. Due to its invisibility, we employ four well-known evaluation indexes to evaluate the clustering results: the Rand Index [127], the Jaccard index [56], the Adjust Rand Index [64], and the F-measure [85]. The data sets in this subsection are all taken from UCI [40]. In order to make the result more persuasive, we select different size, dimension and cluster number to verify our method. All the data sets used in this set of experiments can be briefly summarized in Table 2.1. The comparison results of k-means, spectral clustering, FMST, cciMST and psr-SMT are shown in the following Table 2.2 to Table 2.6.

Using the 15 data sets in Table 2.1 and six algorithms (k-means, spectral clus-

(a) k-means (b) spectral clustering (c) zhong-MST (d) cciMST (e) psr-MST (f) **scaled-MST**

Figure 2.4: clustering results of (a)k-means, (b)spectral clustering, (c)FMST, (d)cciMST, (e)PSR-MST, (f)**scaled-MST**

tering, FMST, cciMST, PSR-MST and scaled-MST-based clustering), we compute the four evaluation index values (the Rand Index, the Adjust Rand Index, the Jaccard index and the F-measure). We compare the results in different methods. "cnae-9" contains 1080 documents of the text business descriptions of Brazilian companies classified into 9 clusters. From the six tables, we know that our method has the best result in terms of Rand Index. "dermatology" aims to determine the type of Eryhemato-Squamous Disease. It contains 33 linear value and one nominal. There are missing values in this data set. "energy-efficiency-y1" accesses the heating load requirements of buildings [144]. All of the methods can perform well on this data set. "fertility" are semen samples provided by 100 volunteers [52]. It is obvious that our method has the best result on this data set. "movement-libras" contains 15 classes of 24 instances each, where each class references to a hand movement type in LIBRAS. PSR-MST performs better on this data set. "nursery" is a large data set, which contains 12958 instances and can be classified to 4 classes. cciMST cannot compute the MST because of the out of memory of large data sets. Our method can perform well on this data set. "ozone-eighthr" and

Table 2.1: Description of data sets.

Data Name	Size	Attribute	Class
cnae-9	1,080	857	9
connectionist	528	10	11
dermatology	366	35	6
energy	768	8	37
fertility	100	10	2
movement-libras	270	91	15
nursery	12,958	9	4
ozone-eighthr	2,534	73	2
ozone-onehr	2,536	73	2
page-blocks	5,473	11	5
plant-species	1,600	64	100
semeion	1,593	257	10
soybean-large	307	36	19
thyroid	3,163	26	2
wilt	4,839	6	2

"ozone-onehr" are two ground ozone level data sets. It is quite evident that our method is much better than the other five methods. In the case of "page-block" data set, FMST is weak in processing this data set because of the existence of empty clusters for sparse data sets. The result of our method is better in terms of Rand, Jaccard and F-measure. However, FMST can perform well on "wilt" on four indexes.

Totally, in terms of Rand Index, our scaled-MST-based clustering has the best performance on 7 data sets. PSR-MST clustering achieves the best evaluation index value on 6 data sets. Spectral clustering can perform well in terms of Adjust Rand Index on 8 data sets.

In order to make the results more clearly, we plot the index value curves by taking the data sets as x-axis and index value as y-axis. The results are shown in Figure 2.5–Figure 2.8. From the observation above, we know that different methods are suitable for different distribution data sets. Our scaled-MST-based clustering method has a better performance on most of the data sets. Besides, our method works well on other data sets.

Table 2.2: Clustering results of k-means clustering in different data sets

Data Name	Rand Index	Adjust Rand Index	Jaccard Index	Fmeasure
cnae-9	0.8384	0.3114	0.2509	0.2859
connectionist	0.8624	**0.2063**	**0.1642**	0.1556
dermatology	0.6959	0.0258	0.1200	0.0978
energy	0.9344	0.0915	0.0656	0.0398
fertility	0.5533	0.0633	0.4964	0.5050
movement-libras	0.9149	0.3480	0.2447	0.2610
nursery	0.6184	0.0675	0.1969	0.1798
ozone-eighthr	0.5269	-0.0311	0.5043	0.5088
ozone-onehr	0.5383	-0.019	0.5286	0.5306
page-blocks	0.6882	0.0123	0.675	0.7207
plant-species	0.9893	0.4902	0.3293	0.3733
semeion	0.8901	**0.4064**	0.3051	0.3259
soybean-large	0.9205	0.4085	0.2904	0.2840
thyroid	0.7350	-0.0547	0.7321	0.7559
wilt	0.5002	-0.0060	0.4744	0.4752

Table 2.3: Clustering results of spectral clustering in different data sets

Data Name	Rand Index	Adjust Rand Index	Jaccard Index	Fmeasure
cnae-9	0.8599	**0.4498**	**0.3568**	**0.4563**
connectionist	0.8163	0.1625	0.1482	0.1609
dermatology	0.7296	**0.169**	**0.2039**	0.1999
energy	0.9322	0.0860	0.0634	0.0382
fertility	0.5679	0.0795	0.5123	0.5227
movement-libras	0.9180	**0.3765**	**0.2659**	**0.2897**
nursery	0.6124	0.0713	0.2080	0.1955
ozone-eighthr	0.5193	**0.0224**	0.4867	0.4885
ozone-onehr	0.5147	**0.0109**	0.4997	0.5002
page-blocks	0.4480	**0.0744**	0.3532	0.3479
plant-species	0.9908	**0.5449**	**0.3788**	**0.4255**
semeion	0.8757	0.4061	**0.3111**	0.3612
soybean-large	**0.9292**	**0.4709**	**0.3403**	**0.3398**
thyroid	0.5412	-0.0415	0.5271	0.5317
wilt	0.5047	-0.0138	0.4807	0.4821

2.4.3 Results on image

To further checkout the performance of our clustering algorithm, we implement our method on color image segmentation. The recognized appraisement

Table 2.4: clustering results of FMST-based clustering in different data sets

Data Name	Rand Index	Adjust Rand Index	Jaccard Index	Fmeasure
cnae-9	0.1615	0.0001	0.1095	0.3104
connectionist	0.4660	0.0215	0.094 0	**0.1962**
dermatology	0.4468	0.0021	0.1759	**0.2654**
energy	0.9129	**0.1806**	**0.1271**	**0.1182**
fertility	0.7715	-0.0164	0.7710	0.8661
movement-libras	0.8365	0.1846	0.1482	0.1914
nursery	0.3523	0.0056	0.311	0.5204
ozone-eighthr	0.8748	-0.0070	0.8747	0.9309
ozone-onehr	0.9344	-0.0085	0.9344	0.9613
page-blocks	-	-	-	-
plant-species	0.8603	0.0793	0.0501	0.1772
semeion	0.8165	0.3114	0.2532	0.3418
soybean-large	0.8090	0.2697	0.2178	0.3164
thyroid	0.8773	**0.0694**	0.8762	0.9130
wilt	**0.8931**	**0.0193**	**0.8929**	**0.9410**

Table 2.5: clustering results of cciMST-based clustering in different data sets

Data Name	Rand Index	Adjust Rand Index	Jaccard Index	Fmeasure
cnae-9	0.1247	0.0000	0.1101	0.3260
connectionist	0.8053	0.1030	0.1140	0.1128
dermatology	0.6537	0.0960	0.1862	0.2027
energy	0.9321	0.1024	0.0731	0.0468
fertility	0.5533	-0.0674	0.5237	0.5459
movement-libras	0.8682	0.2711	0.2003	0.2587
nursery	-	-	-	
ozone-eighthr	0.3175	-0.0304	0.2532	0.2459
ozone-onehr	0.6288	-0.0164	0.6226	0.6287
page-blocks	0.6600	0.0067	0.6429	0.6808
plant-species	0.9299	0.1415	0.0845	0.2045
semeion	0.8184	0.3651	0.2933	**0.4312**
soybean-large	0.9134	0.3998	0.2875	0.2890
thyroid	0.5316	0.0440	0.5043	0.5055
wilt	0.5703	-0.0471	0.5570	0.5652

method is to visually observe the image segmentation results. As an important image processing step, image segmentation aims to separate an image into valuable components. Images taken from a camcorder are stored in RGB (Red, Green,

Table 2.6: clustering results of PSR-MST-based clustering in different data sets

Data Name	Rand Index	Adjust Rand Index	Jaccard Index	Fmeasure
cnae-9	0.6418	0.1115	0.1514	0.2390
connectionist	**0.9171**	0.1917	0.1198	0.1054
dermatology	**0.8149**	0.1501	0.1086	0.0963
energy	**0.9475**	0.0260	0.0195	0.0084
fertility	0.3465	0.0078	0.2142	0.2024
movement-libras	**0.9412**	0.3595	0.2400	0.2213
nursery	0.3175	0.0000	0.3175	0.5635
ozone-eighthr	0.1205	0.0001	0.0027	0.0026
ozone-onehr	0.0583	-0.0000	0.0026	0.0025
page-blocks	0.2104	0.0057	0.0285	0.0271
plant-species	**0.9911**	0.3868	0.2430	0.2250
semeion	**0.9057**	0.1832	0.1176	0.1002
soybean-large	0.9176	0.446	0.3251	0.3377
thyroid	0.0963	0.0009	0.0060	0.0059
wilt	0.1124	0.0005	0.0126	0.0121

Table 2.7: clustering results of our method in different data set

Data Name	Rand Index	Adjust Rand Index	Jaccard Index	Fmeasure
cnae-9	**0.8903**	0.0304	0.0190	0.0146
connectionist	0.9128	0.0780	0.0467	0.0387
dermatology	0.8038	0.0361	0.0257	0.0215
energy	0.9225	0.1386	0.0985	0.0758
fertility	**0.8182**	**0.2948**	**0.8077**	**0.8832**
movement-libras	0.9367	0.1189	0.0718	0.0545
nursery	**0.7620**	**0.5426**	**0.5716**	**0.756**
ozone-eighthr	**0.8775**	-0.0043	**0.8775**	**0.9341**
ozone-onehr	**0.9396**	-0.0043	**0.9396**	**0.9668**
page-blocks	**0.8096**	-0.0007	**0.8095**	**0.8993**
plant-species	0.9910	0.1925	0.1080	0.0905
semeion	0.9018	0.0279	0.0160	0.0146
soybean-large	0.9203	0.1667	0.1030	0.0879
thyroid	**0.9056**	0.0284	**0.9054**	**0.9484**
wilt	0.8862	-0.0112	0.8861	0.9341

Blue) color space. To get high dimensional data sets of image features, we consider converting the RGB color space into HSV (Hue, Saturation, Value) color space. Then we split the Hue into 10 bins, the Saturation into 10 bins and the Value into

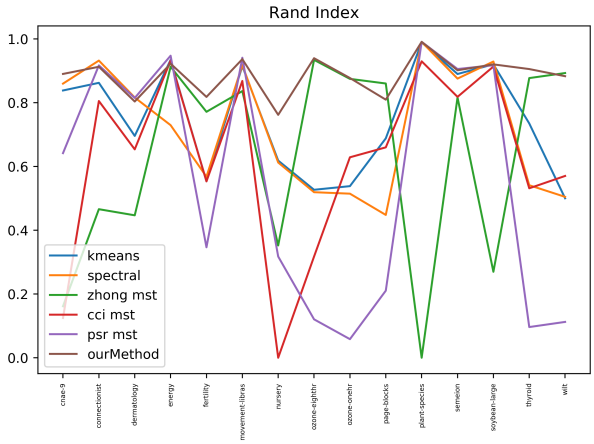


Figure 2.5: Rand index value curves for different algorithms.

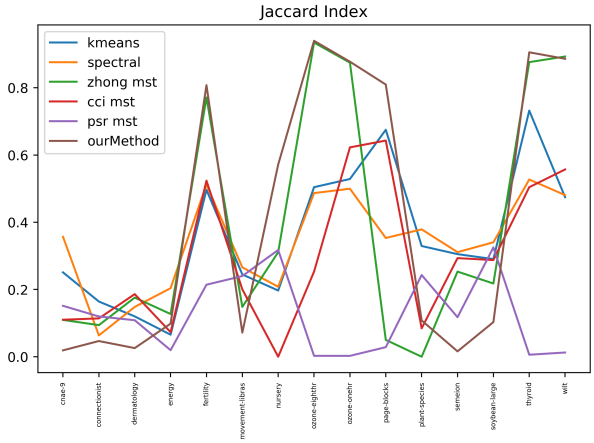


Figure 2.6: Jaccard index value curves for different algorithms.

100 bins. Then each color can be represented by a size of $10 \times 10 \times 100$ array so the dimension of image feature vectors we generate from image is 10000. Every pixel in the image can be denoted by these three bins. We employ a moving window with size $N \times N$ and move this window step by a parameter $Nhop$. We construct a color histogram for the moving window by using the number of pixels in the color bins. Then we use the values in color histogram divided these 10000 numbers as the values of feature vector. The original image can be split into several regions

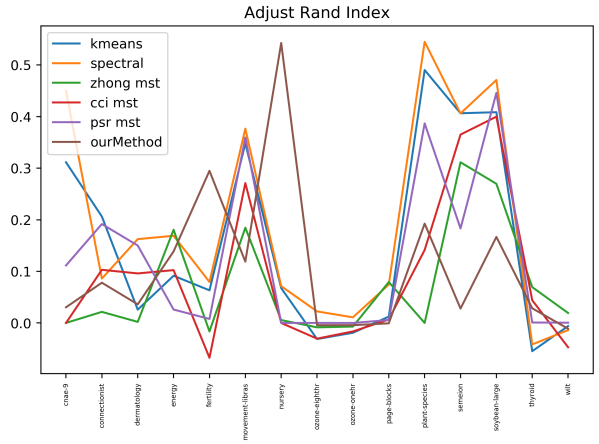


Figure 2.7: Adjust Rand index value curves for different algorithms.

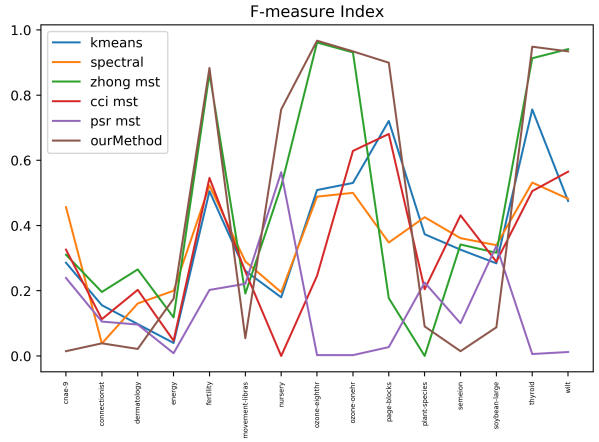


Figure 2.8: F-measure index value curves for different algorithms.

by the $N \times N$window. The size of the image we generate is depend on the value of N and $Nhop$:

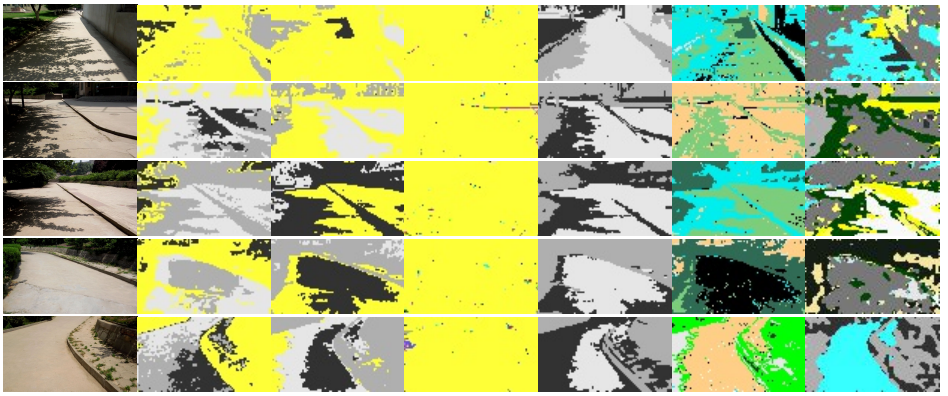
$$blockrows = (height - N)/Nhop + 1 \tag{2.10}$$

$$blockcols = (width - N)/Nhop + 1 \tag{2.11}$$

where *blockrows* is the height of the image we generate, *blockcols* is the width is the width of image we generate, *height* and *width* are the original image size. After extracting image features, we can get a highly sparse feature vector with 10000 dimension. To read data sets more efficiently, we just read the non-zero value and the corresponding index.

Outdoor environments are less structured. We choose five outdoor scenes to obtain image feature vectors. The original size of the images is 1280×720. In this case, we set $N = 21$, $Nhop = 14$. Then we can obtain 90×50 image feature vectors with 10000 dimension. The original images are demonstrated in the first column of Figure 2.9. As shown in the image, there are trees and roads in the pictures. The result of k-means and spectral clustering are illustrated in the second column and the third column separately. The number of clusters is set to be 4. In the case of the fifth picture, k-means oversegments the road into 3 clusters. However, if we set the cluster number to be 3, the road and tree cannot be separated well. In the case of picture 1, it is not clear to see the border of the tree and the road. We can see the road in picture 2 and picture 3. Picture 4 and picture 5 are oversegmented by spectral clustering. The result of FMST clustering is shown in the middle column. It is not clearly to identify different objects because of the high dimension. As shown in the fifth column and the sixth column of Figure 2.9, cciMST and PSR-MST have better performance. They can discover the border of road in these pictures. In our case, the blue area in the first picture and the last picture stands for the road in the original picture. The gray area in picture 2 and picture 4 are the roads in original pictures. The road in picture 3 is illustrated by white. Although some noises exist in the results, our scaled-MST-based clustering can identify the border of different objects.

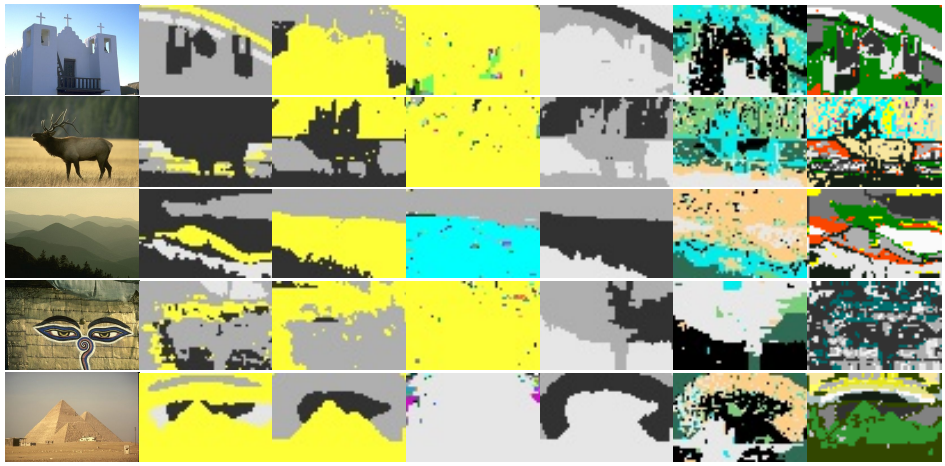
Furthermore, we verify our method on Berkeley Segmentation data sets, which is composed of natural images. These images contain complex layouts of different textures and various shapes. Therefore, the segmentation on these images is a challenging problem. The original size of the images is 481×321. In this case, we set $N = 12$, $Nhop = 8$. Then we can obtain 59×39 image feature vectors with 10000 dimension. As shown in Figure 2.10, we choose five pictures to checkout our method. The original images are demonstrated in the first column. From the middle column, we can see that FMST have no advantages on high dimensional data sets. There is a church building with white wall and



(a) source (b) k-means (c) spectral clustering (d) zhong-MST (e) cciMST (f) psr-MST (g) **scaled-MST**

Figure 2.9: (a) source clustering results of (b)k-means, (c)spectral clustering, (d)FMST, (e)cciMST, (f)PSR-MST, (g)**scaled-MST**

three white crosses on the top in the first picture. For this picture, it is obvious that k-means can discover part of the church and spectral clustering can discover the three crosses. cciMST can only distinguish the church and sky but not the shape of church. PSR-MST and our method can discover all of these. For the second picture, there is a deer on the green grass in front of the forest. k-means can identify part of the deer. Some part of deer and some part of the forest are mixed. The shape of deer can be well identified by spectral clustering, cciMST and our method. However, our method oversegments forest and grass. There are too many noises for PSR-MST. There are several undulating mountains in the third picture. k-means can discover two mountains, while the shape of mountain cannot be shown clearly by spectral clustering, cciMST and PSR-MST. Our method can discover all the shapes of mountains, but oversegments the sky. For the fourth picture, two eyebrows, two eyes and one nose from up to down. It is evident that k-means and spectral clustering can discover the border of face region. cciMST and PSR-MST are weak on this image. Our method can detect the eyebrows, eyes and nose. In the case of last picture, there is a pyramid on a desert in front of the sky. k-means and spectral clustering perform well on separating the pyramid and sky, but mix the pyramid and the desert. cciMST can only find



(a) source (b) k-means (c) spectral clustering (d) zhong-MST (e) cciMST (f) psr-MST (g) **scaled-MST**

Figure 2.10: (a)source, clustering results of (b)k-means, (c)spectral clustering, (d)FMST, (e)cciMST, (f)PSR-MST, (g)**scaled-MST**

the sky. PSR-MST can identify part of the pyramid and others are misclassified to the desert. Our method can detect sky, pyramid and desert well.

Overall, it is obvious that FMST cannot perform well in high dimensional data sets. k-means, spectral clustering and cciMST perform better than FMST, but the results are not better to identify border of different objects. PSR-MST performs well on outdoor pictures, but perform worse on Berkeley data sets. Comparing with these five algorithms, our method performs better (see Figure 2.10). For objects with different colors, our method can always identify the border of the object. However, a shortcoming of our method is that the noise exists in the clustering result, which results in oversegmentation of images. There are some possible reasons for this result. One of the reasons is that the value of parameter *noise_threshold* is a little small in our method. The values of this parameter have a big influence on the result. We can improve the performance by modifying the parameter values. To improve the quality of image segmentation, we can detect noise first and delete them.

Furthermore, we use our image segmentation and integration method on the first set of images. In our case, after running our scaled-MST-based clustering

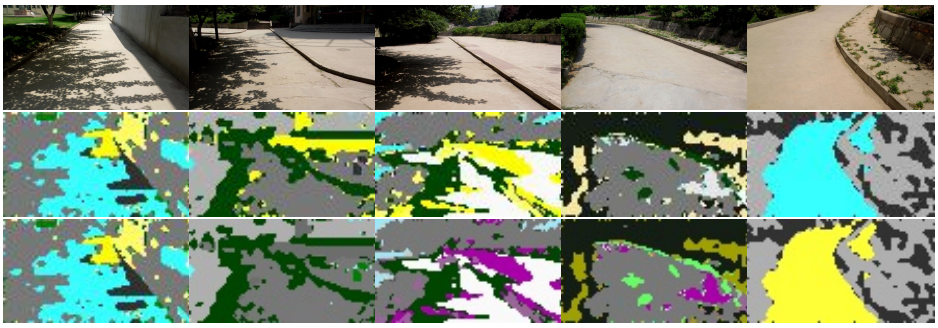


Figure 2.11: The first line : source picture, the second line : clustering results of our method, the last line : results of integration

algorithm, the image is separated into several groups. Before merging the two pictures, we randomly choose one picture as train data and all other four pictures as test data. There are 4 clusters in picture 1 and 8 clusters in picture 2. We construct a cover tree for all of the data in picture 1. Then we employ SVM to predict every clusters in picture 2. We determine whether to merge the clusters in these two pictures according to the conditions in the last section. We repeat these processes until all the pictures are determined. After clustering and merging the similar clusters, the cluster numbers in the five pictures are: 4, 8, 10, 12, 3. We can easily see the results of image integration in Figure 2.11.

2.5 Conclusion

As an important technique in data mining, MST-based clustering algorithm can find clusters with arbitrary shapes comparing with other clustering algorithms. However, traditional MST-based clustering algorithm only considers the Euclidean distance between two points, which has many obvious shortcomings. One of the significant disadvantages is that it may results in an island when removing the inconsistent edges, especially for the data sets with different density. To address this issue, this paper introduces a new method of MST-based clustering, called scaled-MST-based clustering. By considering the scaled distance of an edge, our method can find the longest edge connecting different density clusters. Besides, in order to improve the performance of MST-based clustering, we modify our

scaled-MST-clustering by merging the MST construction and inconsistent edges' detection into one step. Furthermore, our clustering method is applied to image segmentation and integration to recognize the new object in an image. Our contribution is not only improving the quality of clustering results, especially for the different density distribution data sets, but also reducing the time complexity of large image data, which makes the implementation of MST-based clustering more efficient. Three sets of experiments on different dimensional data sets show that the proposed method is credible and effective. In future work, higher performance of MST-based clustering algorithms should be further studied. And we should focus more on the reducing of parameter influence. Much large data sets which cannot fit into memory should be addressed.