A Segmentation Preprocessor

Faster Image Segmentation using Clustering

Jeroen Melchiors

February 16, 1996
Thesis no. 368
Department of Informatics
University of Nijmegen

A Segmentation Preprocessor

Faster Image Segmentation using Clustering

Jeroen Melchiors

Thesis no. 368
University of Nijmegen
Department of Computer Science
February 16, 1996.

This document is prepared using latex

Contents

List of Figures

List of Tables

Preface

The report you have in front of you describes the development of a preprocessor which reduces the time needed to calculate a segmentation of a satellite image. The research project was carried out at the department of Experimental Informatics for Technical Applications (EITA) at University of Nijmegen.

I would like to thank my fellow students and friends for their constructive critisism and useful insights. A special mention in this deserve Erwin Hebing and Eric Boon who listened to me countless times and were always willing to help me along with useful tips and sugesstions. I would also like to thank my supervisor Dr. Th. Schouten for the effort he has put into guiding me through this project.

In the last place I want to thank my parents who have supported me all these years and have given me incentive and the opportunity to achieve this milestone.

Jeroen Melchiors
February 1996.

Chapter 1

Introduction

This report describes the development of a preprocessing phase for segmentation by region growing of images of the earth obtained by Remote Sensing. The objective of the research project is to construct a preprocessor which speeds up the segmentation process compromizing as little as possible the quality [TES94] of the resulting segmentation. The segmentation method used is Schoenmakers' SOIM (Simultaneous Order Independent Merge) [Sch95], which is based on the HSWO (Hierarchical StepWize Optimization) best merge algorithm by Beaulieu & Goldberg [BG89].

In order to construct such a preprocessor, the following questions have to be answered:

1. What information can be extracted from a satellite image.
2. What determines the complexity of the SOIM segmentation algorithm?
3. How can the extracted information be used to speed up the segmentation process?
4. What effect does preprocessing have on the quality of the resulting segmentation?

Answers to these questions will steer the development of a preprocessor for image segmentation.

The rest of this chapter contains an introduction to remote sensing and image processing techniques used in remote sensing.

In chapter 2, the first question is answered and Cluster Analysis is discussed as a way of structuring a data set.

Chapter 3 discusses the implementation of the clustering algorithm used in the preprocessor.

An answer to questions 2 and 3 can be found in chapter 4, where the design of the preprocesor is discussed. An assessment of the effect preprocessing will have on the segmentation process is made.

In chapter 5 the preprocessor's functionality is tested. A method of measuring segmentation quality is discussed. 6 different test cases illustrate the preprocessor's effect on segmentation quality and segmentation speed using different configurations of the preprocessor as well as several different input images.

Chapter 6 contains the conclusions of the research project and possible further work.

1.1 Remote Sensing

Remote sensing gathers information about an object, area or phenomenon without direct contact with that object, area or phenomenon. Examples are images made of the earth's surface by a satellite or airplane and X-ray images of a patient, made in a hospital. The rest of this paper will deal only with remote sensing of the earth's surface. From now on the term "image" will mean image of the earth obtained by remote sensing by a satellite.

1.2 Properties of satellite images

Satellite images are made using sensors scanning the earth's surface. These sensors are sensitive to specific wavelengths. Usually a recording is made using scanners which are sensitive to different frequency bands. Well know satellite platforms are the Landsat and SPOT, for a technical discussion about them see [Sch95] and [She95]. The scanned image is stored as a map of pixels, a pixel being the smallest possible element in the image representing an area on the ground. These pixels have values corresponding to the measurements taken by the scanners aboard the satellite. In case of the Landsat satellite, each pixel has values in 7 spectral bands. These bands are blue, green and red in the visual part of the spectrum and several infrared bands (see table 1.1).

Band number	Spectral range [1]			
1	0.45	to	0.52	blue
2	0.53	to	0.61	green
3	0.63	to	0.69	red
4	0.75	to	0.90	near-IR
5	1.55	to	1.75	mid-IR
6	10.40	to	12.50	thermal-IR
7	2.05	to	2.35	mid-IR

Table 1.1: The spectral bands recorded by the Landsat 7 satellite platform

Different satellite platforms have different ground resolutions and produce different image sizes. For example, the size of an image made by the Landsat satellite is 5760 x 6100 pixels, to store it takes about 250 MB. The ground resolution of a Landsat image is 30 meters, thus every pixel in the image represents an area of 30 by 30 meters. This means that small objects can not be seen in the image and that a pixel may cover several objects on the ground. Such a pixel is called a mixed pixel. A Landsat image of 5760 x 6100 pixels covers an area of 172.8 by 183 kilometers.

1.3 Structuring the information

In order to extract information from the image it has to be processed. A satellite image is a map of pixels of unstructured information. This means that there is no information about which pixels belong to the same field on the ground or where the border between two fields is located. The human eye is capable of deriving this information from the image on sight. The computer is not able to do this in the same way but needs to order the information in some way.

[1] in μm

In order to structure the information several methods have been developed. These all try to recognize objects such as fields, rivers and other infrastructure, as well as their boundaries. An attempt is made to group pixels which are similar according to a formulated criterion. For example a threshold value may be used to decide which pixels are similar and which are not. Several algorithms have been developed to generate a segmentation of an image. Such algorithms divide an image into distinct segments. In the case of an image of an agricultural area, the segments could each represent one agricultural field.

Usually, segmentation is used in a pre-processing stage. The segmented image may be used in a wide range of applications. For instance, a geological survey which tries to track the rate of deforestation in the Amazon rain forest. In this case the segments found in the image can be classified into several groups or classes. These are chosen in such a way that members of a certain class have properties which are of interest and which make classification possible. These classes are often called ground truth classes for the classification, i.e. corresponding to the crops which grow in different agricultural fields. In [geoapp] a discussion of several applications can be found.

1.4 Approaches to image analysis

There are basically two approaches to image analysis. The segmentation approach tries to group pixels of an image into segments. The classification approach classifies all the pixels in the image according to their spectral properties into several predefined classes representing specific features of certain ground cover types, see [Sch83].

1.4.1 Image segmentation

The proces of grouping pixels in an image is called segmentation. The idea is to form segments of pixels which correspond to objects seen in the original image. Segmentation methods can be grouped in 3 categories:

- Edge finding
- Region growing
- Hybrid methods

Edge detection methods

Edge detection methods try to find strong discontinuities in values of adjacent pixels in the satellite image. These discontinuities can for example be found at the border of two adjacent agricultural fields where different crops are grown. The discontinuity is compared to a predefined threshold value, if it is larger, an edge in the image is found. Edges are then joined together, deleting spurious and weaker ones and filling in the gaps, in order to find the boundaries of segments. A well known example of an edge detection method is the one described by Lacroix [Lac88].

Region Growing methods

In contrast to edge detection methods, region growing looks for similarity in neighboring pixels and tries to group them if they satisfy a similarity criterion. In region growing a seed point is chosen and compared with its neighbors. A similarity criterion could state that a pixel can be added to a region if its pixel value(s) do not differ more then a set threshold from the mean pixel value of that region. It is difficult to set a good threshold. If the threshold is chosen too low, fields in the original scene may be split up in several regions. Also a lot of isolated pixels (regions containing only one pixel) may result. If the threshold is set too high, regions may result containing more than one area in the original scene. This is called overgrowing. Haralick & Shapiro [HS85] give a detailed account of different region growing methods.

Hybrid method of segmentation

A hybrid method for segmentation of satellite images has been developed by Schoenmakers [Sch95]. The method is based on a combination of edge detection and region growing. The method of edge detection is a modified version of the one described by Lacroix [Lac88]. With this algorithm "strong edges" in the image are detected. Edge detection does not always produce connected edge segments. A proces called edge following or edge linking will attempt to connect edge fragments by adding pixels to segments. Edge linking produces closed polygons comprised of the found edge fragments. The interior of these polygons may contain more than one area of the original scene. Edge fragments which are not part of a closed polygon are pruned away.

On each closed polygon region growing is performed to find areas which are separated by "weak edges". Region growing is not allowed to pass the boundary of the polygon. The Region growing method used is a modified version of the HSWO (Hierarchical stepwize optimization) algorithm [BG89], and is called SOIM (Simultaneous Order Independent Iterative Merge). The main difference with HSWO is that pairs of segments having the same similarity value are merged simultaneously in the same iteration. In HSWO this takes more than one iteration.

There are several advantages of using this hybrid segmentation method. There is a significant run time improvement and the use of parallelism of region growing stage becomes possible. A clear disadvantage of using the hybrid segmentation method is that the image is divided into polygons of limited size (number of pixels). If a field or other object is larger than the limited polygon size, the field or object is split up and cannot be joined during region growing because region growing may not pass polygon borders. [Sch95] gives a detailed account of advantages and disadvantages of using his hybrid segmentation method.

1.4.2 Image classification or clustering

Image classification is a process in which pixels are catagorized by their spectral values. [Sch83] defined classification as the decision making technique which decides how to catagorize a pixel from its spectral features (values). The goal of classification therefore is to find areas in the image which have a similar *spectral signature*. Image classification is also often called clustering.

During classification, the pixels are labeled. The labels correspond to different spectral classes. In a classification pixels with the same labels may belong to different agricultural fields having the same properties which means that pixels assigned to one spectral class may be scattered all over the image. This in contrast to segmentation techniques where pixels in one segment are confined to be a bounded area in the image, i.e. an agricultural field.

The classification process can be divided into three stages:

1. A learning stage

2. A classification stage

3. An output stage

The learning procedure may be supervised or unsupervised. In supervised learning areas from the image are selected which are representative for the different spectral classes in the image. This approach is used e.g. for training neural networks. In order to be able to calculate the spectral signature for a class, a sufficient number of pixels should be available. Usually 10 to 100 pixels per spectral band in the image is enough. After selection of the learning data spectral signatures of the different classes may be calculated. In unsupervised learning alorithms examine a large number of pixels in the image and divide them into different classes. It is assumed that spectral values in one class should be close together in the measurement space and that these should be well separated from spectral values in other classes. The resulting classes can be compared to the reference data about the area in order to map the found classes to the different ground cover types. A well known algorithm is the ISODATA algorithm [For73, Sch83].

During the classification stage all pixels in the image are assigned to classes using information gathered in the learning stage. Classification is done using a classifier which is a criterion used to decide to which class a pixel can be grouped. This is done by calculating properties of a pixel which may be compared to the signatures of the different classes as found in the learning stage. Maximum likelihood [Sch83] is a widely used classifier and neural network classifiers are also being used more often.

In the output stage, the results of the classification are transformed into a user friendly format and usually made available as a map which may be placed over the original image.

1.4.3 Segmentation versus classification or clustering

In the previous sections image clustering and image segmentation were both discussed as being methods of grouping pixels in an image. There is a distinct difference between the two methods:

1. In image classification or clustering the locations of the pixels in the image is not important. Pixels spread over different fields in the image belonging to the same ground cover class can belong to the same cluster.

2. In image segmentation the pixels in the image are assigned to segments. The pixels in each segment must all be connected or confined to a bounded area. Segments representing fields of the same ground cover class are treated as separate segments.

1.5 Applications

Several projects using remote sensing have been launched by the European Union. These include FIRS (Forest Information from Remote Sensing), TREES (TRopical Ecosystem Environment observations by Satellite) and MARS (Monitoring Agriculture with Remote Sensing techniques) [MR87].

In the MARS project the application of remote sensing collecting agricultural statistics was examined. One reason for starting the project was the poor quality and quantity of agricultural information gathered. This was especially true of the crop estimates. In order to improve the accuracy of these statistics techniques like segmentation and classification can be used to preprocess the image data. This information can be used to distinguish, measure and identify the area of the crops, making it possible to estimate crop production and to do forecasting of crop production.

Chapter 2

Cluster analysis

Cluster analysis tries to structure a set of data. The data set can be very diverse in composition, depending on the problem area. The form of the data can be a mix of ordinal, binary and interval values. In the rest of the discussion only the ordinal data type is considered. This means that data units can be associated to a unique natural number. The data values can range from 1-dimensional to n-dimensional vectors. This is a natural choice because in satellite images, a pixel contains values in several channels. A pixel can therefore be seen as an n-dimensional vector of natural numbers. The dimension n is equal to the number of channels in the recording of the satellite image.

The goal of cluster analysis is to group data units into clusters such that the elements within a cluster have a high degree of association while different clusters are easily distinguished from each other. In order to be able to form clusters in the data set it is necessary to define a "measure of association" between clusters in the data set. The entire data set can then be clustered according to this measure. It is not my intention to give an extensive account of clustering analysis, only the basics of cluster analysis are discussed. For detailed accounts see [For73, Har75].

Because pixels in a satellite image can be seen as a data set of n-dimensional vectors, there are two approaches to clustering of this data set can be taken:

- Clustering in the image space
- Clustering in the measurement space

The image space can be defined as the set of all pixels, or pixel vectors, in a paticular image. The measurement space on the other hand is an n-dimensional space in which the pixel vectors can be placed. In order to cluster data in either one of these approaches, an association measure between data items has to be defined.

2.1 Association measures

Because elements in the data set are of the ordinal type, it is easy to define a meaningful measure of association. A good way of expressing the association between two data elements is to look at their distance. To calculate this distance a distance measure can be defined, usually given in the form of a function. A good distance function should have the properties of a metric.

Let E be the representation of a measurement space and let X, Y and Z be points in E. A distance function D is a metric if and only if it satisfies the following conditions:

1. $D(X,Y) = 0 \Leftrightarrow X = Y$
2. $D(X,Y) \geq 0$ *for all* $X, Y \in E$
3. $D(X,Y) = D(Y,X)$ *for all* $X, Y \in E$
4. $D(X,Z) \leq D(X,Y) + D(Y,Z)$ *for all* $X, Y, Z \in E$

The first property states that X is at zero distance from itself and that two points at zero distance are identical. The second property states that distances are non-negative. The third property demands symmetry by requiring the distance from X to Y to be equal to the distance from Y to X. The fourth property states that the triangle inequality must hold for three points X, Y and Z. It demands that the distance travelled from X to Z directly is less than or equal to the distance travelled from X to Z via Y.

Distance measures between data elements

The Minkowski metric is widely used when defining the distance between two n-dimensional vectors. Let X and Y be n-dimensional vectors: $X = (x_1, ..., x_n)$, $Y = (y_1, ..., y_n)$. Then the Minowski metric for vectors X and Y is given by:

$$D_p(X,Y) = [\sum_{i=1}^{n} |x_i - y_i|^p]^{\frac{1}{p}}. \tag{2.1}$$

where p $\geq$ 1. Choosing different values for p gives various metric distance functions. Choosing p = 1 gives the "city block" metric or L_1 metric:

$$D_1(X,Y) = \sum_{i=1}^{n} |x_i - y_i|. \tag{2.2}$$

The euclidean distance metric or L_2 metric is obtained by taking p = 2:

$$D_2(X,Y) = [\sum_{i=1}^{n} |x_i - y_i|^2]^{\frac{1}{2}}. \tag{2.3}$$

This is the most widely used metric. The L_1 metric is occasionally used in image processing. Metrics with higher values for p are mainly of theoretical interest.

Distance measures between clusters

There are also metric distance measures between clusters or sets of data elements. Several well known measures are minimum, maximum and average distance. Figure 2.1 shows how each of these measures is used. The minimum and maximum distance measure give the minimum and maximum distance in the measurement space between the elements in the two data sets. The L_2 metric, or euclidean distance metric, can be used to calculate the distance between each pair of elements in the two data sets.

The average distance measure between clusters can be defined in two ways. Distances between all pairs of data elements can be averaged giving an average distance. Another way of defining the average distance between clusters is to take the distance between the averages of the elements in the two clusters. This second definition is shown in figure 2.1 and will be used from now on as the average distance measure between clusters.

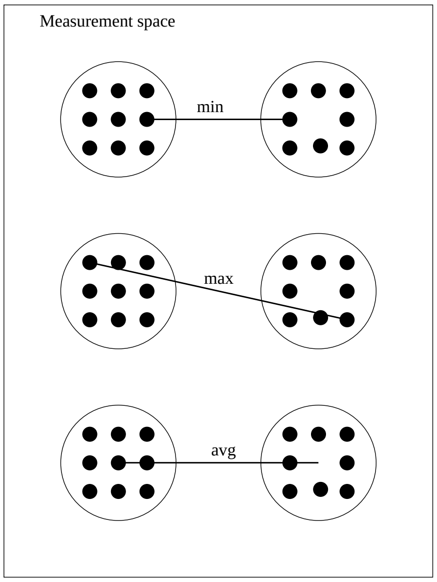


Figure 2.1: Metrics between clusters

2.2 Image space clustering

There are two types of clustering methods that can be used for clustering of data in the image space:

- Hierarchical clustering methods
- Non-hierarchical clustering methods

Depending on the size of the data one of these will be more practical than the other. For an extensive account of clustering methods see [For73]

2.2.1 Hierarchical clustering methods

Using the distance measures described in the previous section, a similarity matrix may be calculated giving the the similarity between all pairs of clusters in the data set.

In hierachical clustering, a clustering tree is built giving a hierarchy in the clustering steps. Initially each data element forms its own cluster and forms its own tree. During each iteration step of the clustering process, the two most similar clusters are merged, decreasing the number of clusters by one. Note that this requires both a distance measure between data elements and a distance measure between clusters to be chosen. The trees belonging to the two most similar clusters are joined. With n data elements, n-1 iterations are performed, and a cluster tree of height n is formed. A special property of a tree belonging to a hierarchical data clustering is that the level in the tree indicates the iteration in which the node was formed. For a simple example of this form of clustering see figure 2.2.

Let the similarity between clusters i and j be denoted as S_{ij}. The value S_{ij} is defined according to the chosen distance measure between clusters. The distance between data elements is then usually expressed as the euclidean distance. Possible distance measures have been described in section 2.1. If S_{ij} is a metric measure then $S_{ij} = S_{ji}$, as is stated in condition 3 of the metric definition. For each pair of clusters i and j the similarity value S_{ij} has to be calculated. This gives $\binom{n}{2} =$

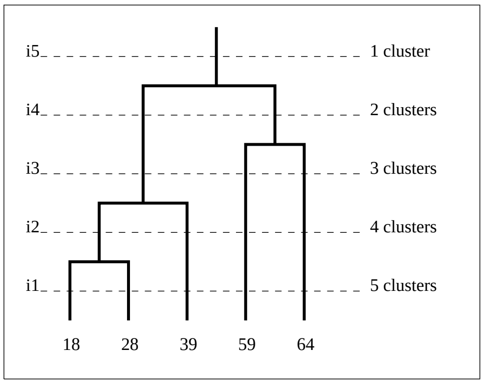


Figure 2.2: Hierachical clustering of dataset (18,28,39,59,64)

$\frac{1}{2}n(n-1)$ similarity values for all possible $\binom{n}{2}$ pairs of n clusters in the data set when using a metric measure.

At the end of the clustering process, one cluster tree is formed. Travelling down the branches, the subsequent merge steps can be seen in reverse order. In order to obtain several clusters, the cluster tree can be traversed at a chosen level. This can be visualized by drawing a line at this level in the tree (figure 2.2). Traversing the tree at level i gives 5 - i + 1 clusters. Choosing i = 1 gives 5 clusters, while choosing i = 5 gives 1 cluster. In general: given a data set containing n elements, traversing the cluster tree of this data set at level i gives n - i + 1 clusters.

Different implementations of hierarchical clustering methods are the stored matrix approach and stored data approach.

- The stored matrix approach builds a similarity matrix containing similarity values between all pairs of clusters i and j in the data set. Only the lower triangular similarity matrix is calculated when using a metric distance measure. Because there are $\binom{n}{2}$ different pairs in a data set of n elements this leads to high memory usage.

- The stored data approach holds the data set in memory and calculates a minimum for the similarity value S_{ij} during each iteration. This takes a lot of comparisons and computations (see [For73] for a detailed analysis).

These two implementations of hierarchical clustering show the "Time - Space - Tradeoff". In this tradeoff, an algorithm is optimized for lowest run time at the cost of memory usage, or low memory usage at the cost of worse run time performance.

The hierarchical clustering steps are as follows:

1. start with each data element as a separate cluster.

2. choose an association measure and calculate the similarity matrix.

3. choose a strategy for updating the matrix elements after merging.

4. repeat:

   - search the most similar clusters and label them p and q.
   - merge clusters p and q.
   - update the similarity matrix by calculating new similarity values between the merged cluster and the other clusters.

- remove the row and column of the cluster with the highest index (p or q) in the matrix.

until there is only one cluster left.

In each step of the clustering, the merged pair of clusters and their similarity value are recorded to later construct the cluster tree.

There are several possible update strategies for bookkeeping in the similarity matrix. An update in the similarity matrix can be seen as linking two clusters. Let the most similar clusters at a certain stage in the clustering process be called p and q. The cluster resulting from the merger of p and q will be called r. Some possible update strategies are minimum linkage, average linkage and maximum linkage where S_{tr} is given the minimum, maximum or average of S_{pr} and S_{qr}. These update strategies use a corresponding distance measure between clusters as described in section 2.1. The update strategy used influences the result of the clustering of the dataset. Generally average linkage gives good result when processing satellite image data.

Using hierarchical clustering is only practical on small data sets. In order to obtain results within a reasonable time, the input dataset should not contain more than 1,000 to 2,000 elements. A satellite image of 45 x 45 pixels could be clustered in a reasonable time using hierachical clustering. Clustering more than a few thousand elements will cost far too much time and or storage space. A 256 x 256 image contains 65535 pixels. During the first iteration of the hierachical clustering, to find the pair of most similar data elements, $\binom{65535}{2}$ or 2.147385×10^9 similarity values have to be calculated. This is a tremendous task and cannot be done using the stored matrix method in memory. To cluster the entire image 65533 more iterations have to be carried out. Hierarchical clustering methods are clearly not practical when handling large data sets as complete satellite images.

2.2.2 Non-Hierarchical clustering

Non-hierarchical clustering is used to cluster the entire data set into a single classification of k clusters. This type of clustering is also known as partition type clustering because the clusters found by a non-hierachical clustering algorithm form a partition of the original data set. A value for the number of clusters k, can be given by the analyst (user) or can be determined by the clustering algorithm [Smi93]. The problem size that can be tackled using non-hierarchical cluster methods is much larger than using hierarchical methods because of the following reasons:

- There is no need for a similarity matrix.
- All data elements are processed sequentially where only a note has to be made to which cluster they are grouped.
- The data set can be processed in parts, possibly parallel.

This means that the problem size using non-hierachical clustering is not limited by memory size, as was the case in hierachical clustering. This is important when handling large sets of data as satellite images.

Most non-hierarchical clustering algorithms consist of the following steps:

1. Choose a set of seed points.
2. Choose a distance measure between data elements (usually euclidean distance is used).
3. Cluster pixels according to the chosen seed points using the distance measure.

4. Determine new seed points based on current clustering result.

5. Goto step 3.

During each iteration of the process, pixels will be assigned to a cluster and possibly switching from one cluster to another because of shifting of the seed points (in step 4). Given enough iterations, the clustering process will stabilize in the sense that pixels no longer switch clusters. At this point, given the initial set of seed points, the optimal clustering is determined. It may be that a local optimum has been found. This is described in [BL94].

The selection of initial seed points influences the outcome of the clustering. Care should be taken that the set of seed points is chosen with enough dissimilarity. The seed points should be "good" candidates for representing the different ground cover classes in the image in order to be able to assign pixels to these classes by clustering.

This clustering method permits parallel processing of the clustering or partitioning step, as well as parallel calculation of new seed points. The data can be chopped into n chunks where n is the number of processors available. Calculation of overall new seed points can be done by combining local results and statistics. Data parallelization over n processors should give an n fold speed up of the clustering process.

2.3 Measurement space clustering

An other way of looking at clustering is to look at the measurement space in which the elements of the data set lie. In a satellite image information is recorded over a large wavelenght range split into several spectral bands or channels. In images from the Landsat satellite each pixel typically has a value in 6 spectral bands (see table 1.1 and [She95]). An example of band in a Landsat image is shown in figure 2.3.

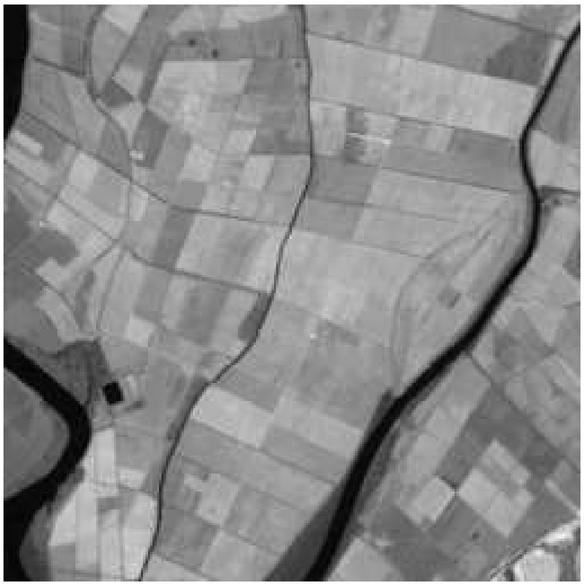


Figure 2.3: Channel 5 of the image suba.lan

This image has size 256 x 256 and is part of the larger subjune image, see figure A.2. Each pixel in the image is represented digitally by 6 values ranging from 0 to 255. These 6 spectral values form a 6-tuple or 6-dimensional vector in the measurement space. Therefore each pixel in the image can be seen as a point in a

6-dimensional cube with sides 256. This cube defines the measurement space for a typical Landsat image. The number of distinct points is in this cube is 256^6. This is the number of possible 6-tuples of spectral values for a pixel. This is much greater than the 256^2 pixels in the suba-image. This means that the cube will be mostly empty. For images containing more spectral bands, this ratio will become worse and more of the cube will be empty. This also holds for full Landsat images of 6000 x 6000 pixels.

Pixels in the image (image space) are mapped to points in the measurement space. It is very likely that these points will form clusters in the measurement space. This is because pixels in the same field or belonging to the same ground cover class will be mapped close to each other in the measurement space. This mapping function does not have an inverse for two reasons:

- More pixels can be mapped to the same point in the measurement space.
- Because there are more points in the measurement space than there are pixels in the image, there are points in the measurement space that do not correspond to pixels in the image.

A possible measure for distance between points in the image space is given by the L_2 measure, the euclidean distance. The euclidean distance beteen two points A and B in n dimensional space is given by:

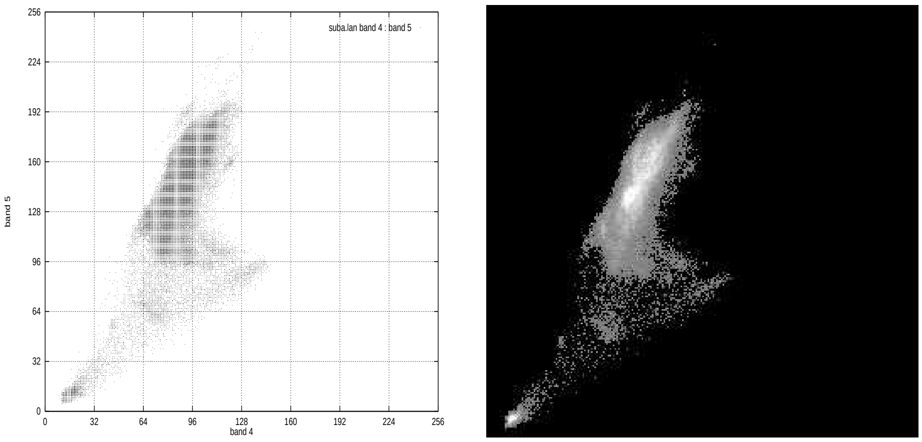
$$d = \sqrt{\sum_{i=1}^{n} |A[i] - B[i]|^2} \tag{2.4}$$

In order to give an impression of the mapping of pixels in the image space to the measurement space, only a 2-dimensional measurement space is considered. In this example band 4 and 5 of the suba image (figure 2.3) were chosen. For this 2-dimensional case, a scatter plot (figure 2.4(a)) and intensity diagram (figure 2.4(b)) are shown. These show that for these 2 bands the spectral values of the pixels in the image are concentrated in a bounded area which is small compared to the entire measurement space. It should be pointed out that the number of pixels in the image is equal to the number of points in the 2-dimensional measurement space in this example.

The intensity diagram gives an indication of how often a certain pixel is mapped to the same point in the measurement space. The bright areas correspond to a large number of pixels in the image all mapped to the same part of the measurement space. If the bright spots in the intensity diagram lie well separated, these can be seen as clusters in the measurement space.

To form a clustering of the measurement space, points P defining the clusters can be chosen corresponding to the bright areas in the intensity diagram. A circle with a certain radius d can be drawn around each point P. All pixels which map to points lying within this circle in the measurement space have a spectral distance $\leq$ d to the point P. The problem now is to choose an appropriate set of points P with corresponding values for d in order to find a clustering of the image.

In the intensity diagram (figure 2.4(b)) a bright spot can be seen in the lower left corner. In this area a starting point P can be chosen around which all pixels mapping to an area within radius d are clustered. This point P may for example be the maximum value, the point in that part of the measurement space to which the largest number of pixels were mapped. The result of this kind of clustering can be seen in figure 2.5. The white pixels belong to the cluster. The other pixels are depicted at half intensity for comparison with the original Landsat image (see figure A.4). In this case one point P was chosen in the measurement space. It was chosen



(a) Scatter plot (b) Intensity diagram

Figure 2.4: Scatter in measurement space of suba.lan in band 4 and 5

Figure 2.5: Segments in the image showing the river (white areas)

as the maximum of the bright spot in the lower left corner of the intensity diagram and d was set at 32. Figure 2.5 shows well-defined regions have been formed in this one cluster in input image. It can also be seen that some pixels in the regions are not clustered, caused by the fact that their spectral values lie outside the definded circle defined by d. To obtain a clustering of the image in more clusters a set of seed points P has to be determined in the measurement space.

2.4 Practical aspects

The clustering technique used as part of a preprocessor as described in the introduction, should produce fast results on large images with great ease of use. In view of these demands hierarchical clustering is not an option as pointed out in section 2.2.1.

Measurement space clustering as described in the previous section is not very practical. It is not easy to locate clusters in the measurement space which is usually 6-dimensional, making the method difficult to use.

The most practical approach for clustering satellite images is non-hierarchical clustering. In the next chapter, the implementation ofa non-hierarchical clustering algorithm is discussed.

Chapter 3

The clustering algorithm

In this chapter a description of the clustering algorithm to be used in the preprocessor is given. This will be followed by some clustering examples will be given to illustrate the working of the clustering algorithm. The chapter ends with remarks about the obtained clustering results in the examples.

3.1 Implementation of the clustering algorithm

The clustering algorithm described in this chapter is a variant on the K-means clustering algorithm. Figure 3.1 shows the steps carried out by the clustering algorithm. In the K-means algorithm, K starting points are chosen so that K clusters may be formed in the data set. While the data elements are assigned to the different clusters, the seed points are continuously updated, reflecting the new cluster means. This process is iterated several times until a stable partition has been found. In the next sections the implementation of the clustering steps will be described. In the clustering algorithm described here, seed points are chosen fixed during an iteration. After all pixels are assigned, new seed points based on the current cluster means are calculated. Seed points are fixed during an iteration for 2 reasons:

- Computational efficiency. Continually recalculating seed points as in K-means slows down the clustering process, the desired clustering speed can not be reached.
- K-means assumes a random distribution of elements in the data set, choosing the first K elements as initial seed points, updating them as pixels are assigned to the clusters. In the case of a data set in the form of image data, the data set is not ordered randomly. The K-means approach is not practical in this case.

3.1.1 Read image data

The satellite data used in this research project is stored in disk files in the ERDAS format [ERD91]. The file consists of 2 parts, a header and the actual image data. The header has the following format:

```
typedef struct lan_header LanHeader;
typedef struct lan_header {
 Byte        hdword[6];   /* pos 1-6     */
 Ushort       ipack;       /* pos 7-8     */
 Ushort       nbands;      /* pos 9-10    */
 Byte        unused1[6];  /* pos 11-16   */
```

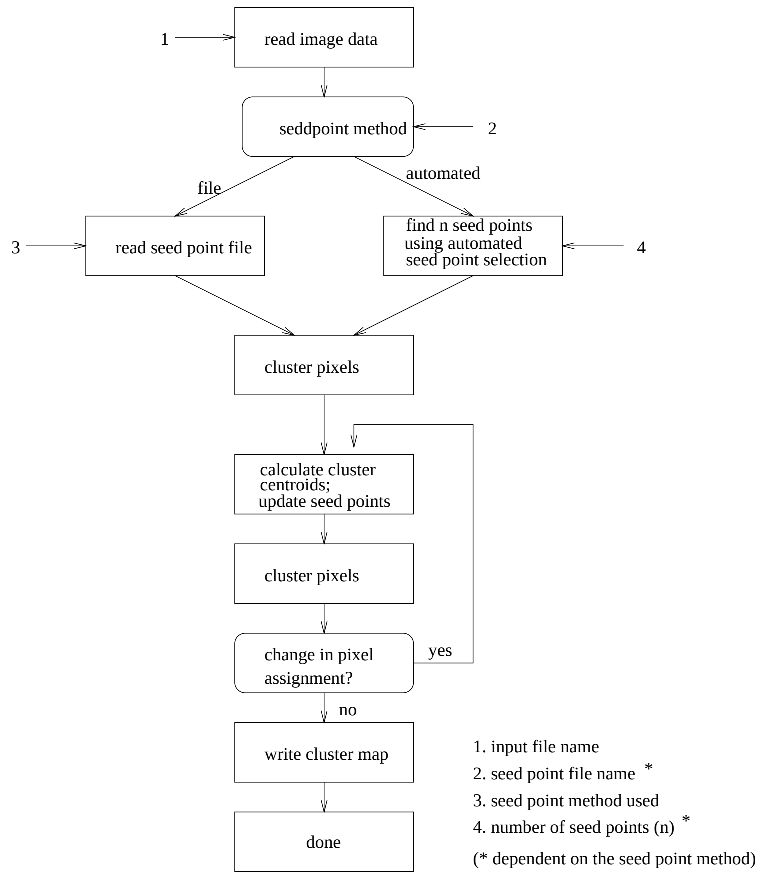


Figure 3.1: Steps of the clustering algorithm

```
 float       cols;        /* pos 17-20   */
 float       rows;        /* pos 21-24   */
 float       rx;          /* pos 25-28   */
 float       ry;          /* pos 29-32   */
 Byte        dummy[88];   /* pos 33-120  */
 float       xcell;       /* pos 121-124 */
 float       ycell;       /* pos 125-128 */
} Header;
```

In this header the following fields are used for reading and writing files in the ERDAS format:

- **hdword** indicates the erdas-version,
- **ipack** indicates the number of bytes per pixel (usually 1),
- **nbands** gives the number of channels in the image,
- **cols** and **rows** give the number of columns and rows of the image.

The implementation of the preprocessor discussed in this report uses the ERDAS-74 format. When using data in an other format, i.e. ERDAS-72, the data should be converted before it can be processed by the preprocessor. A convesion program called erdas72to74 was developed for converting files in the earlier ERDAS-72 format to ERDAS-74.

The image data is stored in BIL (Band InterLeaved) format in the following order: row 1, channel 1, column 1 ... column n, row 2, ... To store the data in memory, a data structure has to be created and memory allocated. Because of the application, a clustering algorithm, it is efficient to be able to address a pixel by its x and y coordinates as in the original image. The coordinate system used in this implementation has its origin (0,0) at the top left and (rows,cols) at the lower righthand corner. A data structure to handle this storage need is shown in figure 3.2.

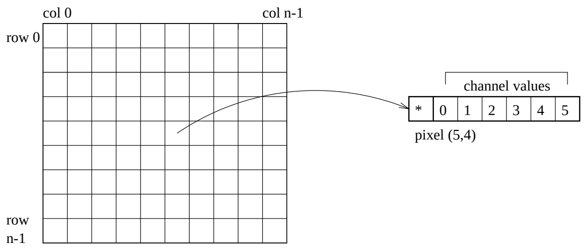


Figure 3.2: Storage of image data in memory

The advantage to this storage method is that individual pixels are easily accessed by subscription in an array. In the implementation, access to the data is achieved by pointers. A piece of memory large enough to hold the data has to be allocated. A single pointer accesses a row of pixels, a double pointer can access a pixel (vector). A three double pointer can access the pixel's individual channel values.

Aside from the above mentioned advantage of easy access to individual pixels by array subscription, there is the advantage of the usage of small memory blocks. In contrast to the allocation of a one-dimensional list of size **cols** x **rows**, in a contiguous block of memory, in the implementation described in this report, a much smaller memory block of size **rows** of pointers is allocated. For each row a block of size **cols** of pointers to a block of **nbands** cells containing the actual data is allocated.

The advantage of using small data blocks is that this will lead to efficient memory usage and will not cause allocation errors in a badly fragmented memory. While this implementation uses pointers to create the data structure, assignment in this data structure is as easy as array subscription.

Disadvantages are that a lot of memory allocations have to be done in order to build the pointer map and that a large number of pointers are used.

For the manipulation of ERDAS files the department has implemented a module erdasio. It contains basic functions for reading and writing of ERDAS files as well as functions for manipulation of the ERDAS header. Functions from this module are used for handling the ERDAS files.

Reading of the image data is done in three steps:

1. read the file header and determine the dimensions of the image data: **rows**, **cols** and **nbands**
2. allocate memory to accomodate the actual image data
3. read the image data and store it in memory

Read file header

The file containing the image data is opened using the **fopen** function. The header of the ERDAS file is read using the function

```
read_erdas_header(FILE *fp, int *elements, int *lines, int *channels)
```

Here fp points to the file containing the image data. **read_erdas_header** returns the values of **cols**, **rows** and **nbands** via the parameters *elements*, *lines* and *channels*. It is important that the input file is in the ERDAS-74 format. If it is not, a conversion should be done.

Allocate memory

After the header has been read, memory can be allocated using the function

```
Byte ***alloc_memory(int lines, int elements, int channels)
```

alloc_memory uses the malloc() function to allocate individual memory blocks. The return value is the pointer to a memory block if memory allocation was successful, NULL otherwise.

Let im be a pointer to the allocated memory block where image data is to be stored. The data structure may be visualized as a 3-dimensional array. Data access is achieved by subscription. im[i] selects the i^{th} row of the data structure, im[i][j] selects pixel (i,j) and im[i][j][k] points to channel value k of pixel (i,j).

Read image data and store it in memory

Image data in the ERDAS file is stored in BIL format. This means that the order of storage in the file is **rows**, **nbands**, **cols**. This differs from the memory storage order of **rows**, **cols**, **nbands** used in the implementation of the clustering algorithm. Therefore the routines for reading image data in the module erdasio cannot be used. These routines assume the BIL order. The function

```
int read_image_data(Byte ***im, int rows, int cols, int nbands,
                      FILE *fp)
```

reads the data and stores it in the allocated memory block im. Reading of a block of bytes from a file can be done using the standard C function **fread**. The data in the ERDAS file is stored in BIL order and is also read in this order. Because the order of storage in memory (figure 3.2) is not band-interleaved but row, column, channel order, the data is reordered befored it is stored in memory.

3.1.2 Determine seed points

In order to find clusters in the data, seed points or cluster definitions have to be determined. In the implementation described in this report, the number of clusters is always fixed, given in by the user. A cluster is defined by a single seed point. Seed points can be determined in two ways:

- Automated, chosen by an algorithm
- Manually, given by the user.

Automated selection

Automated selection of seed points is done by the function

```
Byte **find_seedpoints(int nr_of_seedpoints, int *start_vector,
                       int min_dist)
```

This function returns a set of pixel-vectors selected from the image data. The returned set will contain image pixels which lie at least **min_dist** apart. The start vector can be used to give an initial positioning for the seed points in the measurement space. The distance is assumed to be the euclidean distance as described in section 2.1. Figure 3.3 shows the subsequent selection of 3 seed points. The order is indicated by overlap of the circles with radius d, at least **min_dist**. In this example the number of channels in the image is chosen to be 2, giving a 2-dimensional measurement space. If the number of channels is n, then the seed points will be enveloped by n-dimensional spheres. From the view point of computation speed,

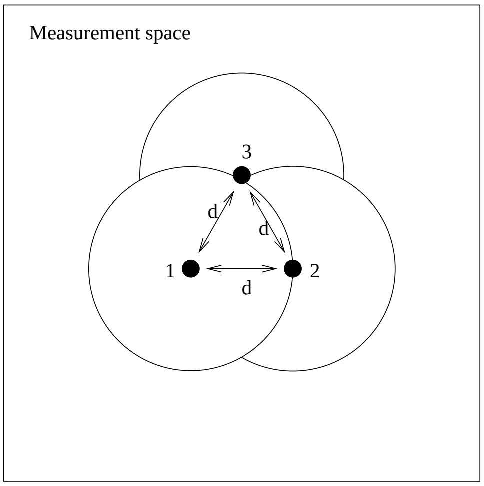


Figure 3.3: Automated selection of seed points

the image data is scanned **nr_of_seed points** times, adding a new seed point to the set based on the seed points already selected as illustrated in figure 3.3. The new seed point will be the pixel closest to all previously selected seed points, lying at least at min_dist from all of them.

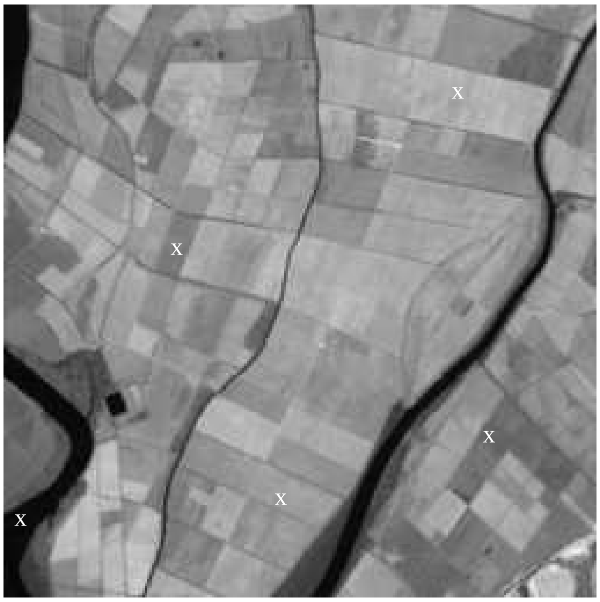
The algorithm does not try to find the best seed point distribution as this would take too much computation time. The setting of the parameter **min_dist** has great influence on the distribution of the seed points. If the **min_dist** selected is too high, at some point there will be no more candidates for the next seed point. The measurement space will be covered by the circles of the previously found seed points. Choosing too low a setting for min_dist will not give a good scatter of seed points over the measurement space. The following values for the parameters were used in the examples in this report and gave quite good results:

- start_vector set to centroid of entire image data set,
- min_dist set to standard deviation.

Note that these parameters can be set using the command line options of the clustering algorithm.

Manual selection

Manual selection relies on the choices made by the user. In figure 3.4 a selection of 5 seedpoints in the suba.lan image is shown. Each seed point is marked with an "X". The coordinate information is also given, it is generated using XV: an image viewer for X-windows developed by John Bradley. The seedpoints can be stored in a file in the form shown in figure 3.4.



```
5 seedpoints suba.lan
  5, 222 =  13, 13, 13  #0d0d0d  (  0   0   5 HSV)  [    0,    0]
 72, 104 = 124,124,124  #7c7c7c  (  0   0  48 HSV)  [    0,    0]
190,  33 = 179,179,179  #b3b3b3  (  0   0  70 HSV)  [    0,    0]
122, 213 = 162,162,162  #a2a2a2  (  0   0  63 HSV)  [    0,    0]
204, 202 = 106,106,106  #6a6a6a  (  0   0  41 HSV)  [    0,    0]
```

Figure 3.4: Manual selection of seedpoints using XV

The function

```
Byte **read_seedpoints(char *coord_file)
```

can read a file in the format of figure 3.4 and returns a list of pixel-vectors corresponding to coordinates specified in **coord_file**. The first two numbers on a line in the **coord_file** indicate the x and y position of the pixel, Note that XV uses an (x,y) coordinate system where as in the cluster algorithm implementation data is stored in an (y,x) order.

This method of seed point selection relies on the good judgement of the user. The composition of the set of seed points will influence the clustering result. This method of manual selection of seed points should only be used if no ground truth data set is avalilable for the set of satellite images.

If a ground truth data set is available, the average intensities of the ground cover classes, defined in the ground truth data, can be used as seed points. Of the Portugal images, [oaHSM95], the ground truth data set is avalilable. Table 3.1shows 10 ground cover types characterized by their mean Landsat-TM intensities per channel. A file in the form of table 3.1 can be read and processed by the clustering algorithm. Each line of the file should contain the average channel values of one ground cover class.

3.1.3 Cluster the data

Now that seed points are determined, the actual clustering of the image data can take place. The clustering result is stored in the **cluster_map**, a 2-dimensional array of integers of size **rows** x **cols**. In each cell of the array, the number of the cluster to which that pixel is assigned is stored.

class	1	2	3	4	5	6
sand	149	91	143	130	236	145
soil	92	45	63	64	120	64
water	66	27	28	20	20	9
wheat	91	46	69	92	121	55
barley	92	48	77	101	161	71
maize	80	37	43	125	110	43
rice	70	31	34	347	34	14
vine	104	54	75	107	161	84
grass	94	48	75	89	149	65
forest	76	34	44	81	98	41

Table 3.1: Ground cover classes with mean Landsat intensities per channel

The **cluster_map** is a pointer data structure created by the function

```
int **create_clustermap(int rows, int cols)
```

This is done in a way similar to the allocation of the image data. The cluster map is initialized with 0 in each cell.

As can be seen in figure 3.1, after the seed points are determined, the pixels are clustered. This will give the initial clustering or partition of the image data according to the given set of seed points. The function

```
cluster_pixels(Byte ***im, int **cluster_map, int **seed_points)
```

will cluster all pixels in the image data to the cluster with the nearest seed point. Distances are calculated using the euclidean distance metric. Results are stored in the cluster map.

After this initial partition is determined, the algorithm enters a loop of updating the set of seed points and clustering the pixels according to this new set of seed points. In this process pixels may switch from one cluster to another. The final clustering result is obtained when no more switching takes place. The process has then stabilized according to the given initial set of seed points. A cluster map of the clustering result is then available. In the implementation a user can select the number of iterations desired. If fewer iterations are necessary to obtain a stable partition the loop is terminated prematurely.

The seed points are updated by setting the seed point of each cluster to the cluster centroid according to the clustering result obtained in the previous iteration.

3.1.4 Give the clustering result in a useful format

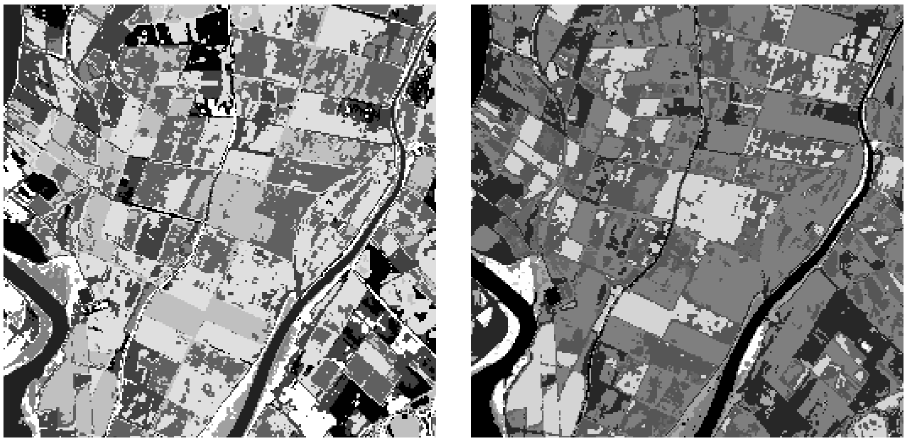
The format in which the clustering result is presented is in the form of a cluster map of the satellite image with a numbering of the pixels according to the cluster to which they are assigned. The cluster map is stored as an ERDAS-74 file with only one channel. This file can be used in later steps of the preprocessor or it can be viewed using an image viewer capable of displaying ERDAS-files.

3.2 Clustering examples

In this section the clustering algorithm is applied to two satellite images: suba.lan and bern.lan, given in appendix A. In the clustering results shown in the different figures in the next sections each cluster is indicated by a distinct gray level.

3.2.1 Clustering of suba.lan

For the clustering of suba.lan, the Portugal ground truth data, table 3.1, can be used. This gives a partition in 10 clusters. The initial partition using this set of seed points is shown in figure 3.5(a). Using automated seed point selection gives the result of figure 3.5(b), also in 10 clusters. There is no correspondence between the gray level values assigned to the clusters using manual seed point selection and those assigned to the clusters using automated seed point selection.



(a) using the Portugal ground truth classes

(b) using automated seed point selection

Figure 3.5: Initial partitions in 10 clusters made of the Suba image

The final clustering results are shown in figure 3.6(a) and 3.6(b). Table 3.2 shows the distribution of the pixels over the different clusters. The cluster numbers of manual seed selection do not correspond to those using automated seed point selection. When comparing the distribution obtained using the different seed point selection methods the following can be seen:

- In manual clustering, cluster 7 was positioned wrongly in the measurement space during the clustering process causing all pixels initially assigned to it to move to other clusters.
- Final cluster sizes are fairly equal in the manual and automated case (appart from cluster 7).
- There are more small clusters using manual seed point selection. In automated seed point selection the pixels initially assigned to cluster 7 had to be redistributed over the other clusters giving larger clusters.

If the clustering results are viewed together with the clustering distribution in table 3.2, it becomes clear that in the case of automated seed point selection the initial clustering result needs a lot more work than the one using the Portugal ground truth set. This was reflected in the number of iterations necessary to obtain stable partitions: 15 to 35. The final clustering results are fairly comparable, keeping in mind that the clustering result using automated seed point selection effectively only contains 9 clusters.

(a) using the Portugal ground truth classes

(b) using automated seed point selection

Figure 3.6: Final partitions in 10 clusters made of the suba image

3.2.2 Clustering of bern.lan

Bern.lan was clustered into 5 clusters. Because no ground truth data was available, automated seed point selection was used. It took 9 iterations to obtain a stable partition. Going from the initial to the final partition a fair amount of pixels were reassigned. The initial partition is shown in figure 3.7(a). Even though a lot of pixels were reassigned during clustering, this does not show clearly in the final partition, figure 3.7(b).

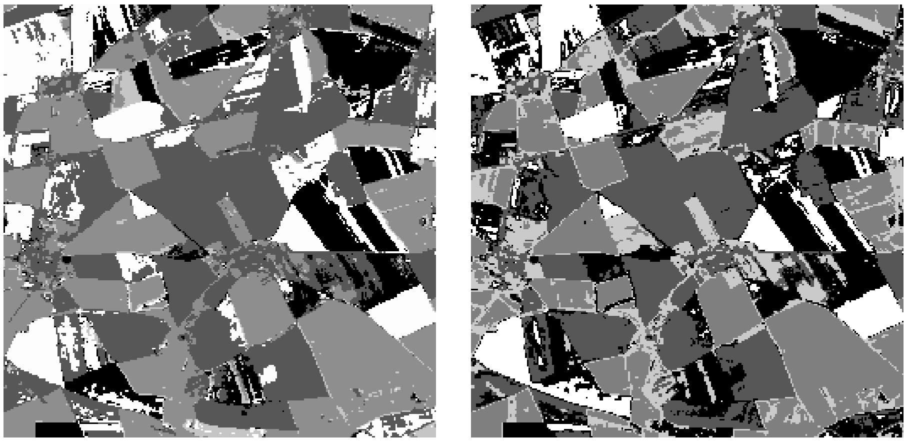
3.3 Concluding remarks

The clustering algorithm described in this chapter is a fast non-hierarchical algorithm for partitioning the pixels in a satellite image into a pre-determined number of clusters, chosen by the user.

Initial seed points for the clusters can be given by the user or generated au-

Cluster #	Initial partitions		Final partitions	
	manual	automated	manual	automated
0	80	13010	476	16823
1	4644	22704	4948	9603
2	3040	9760	2819	4240
3	9643	8891	7229	11420
4	12845	1527	16225	2083
5	1474	898	1778	2820
6	995	1126	2082	1231
7	9452	3494	10892	1
8	18783	719	14091	2260
9	4580	3407	4996	15169

Table 3.2: suba.lan: pixel assignment during the clustering process



(a) initial partition (b) final partition

Figure 3.7: Partition in 5 clusters made of the Bern image

Cluster #	Initial partition	Final partition
0	7984	15454
1	23120	18856
2	21201	15148
3	2096	8872
4	11135	7206

Table 3.3: bern.lan: pixel assignment during the clustering process

tomatically. In both cases care should be taken that the set of initial seed points can "represent" the different ground cover classes in the image data. Section 3.2.1 shows the effect the selection of the initial set of seed points can have on the final clustering result.

Looking at the examples of the previous sections, the clustering results give a reasonable representation of the ground coverage of the original satellite images as shown in appendix A.

Chapter 4

Preprocessing

In this chapter the development of the preprocessor is discussed. The chapter starts with the properties of the best merge algorithm. Then the role of preprocessing can play in the segmentation process is discussed. This is followed by some objectives of using a preprocessor as part of the segmentation process. Then the design of the preprocessor is discussed. The chapter concludes with an assessment of the effect preprocessing can have on the segmentation result.

4.1 Properties of the Best Merge algorithm

The best merge algorithm, as used in this research project is based on the SOIM (Simultaneous Order Independent Merge) algorithm designed by Schoenmakers [Sch95]. The algorithm does region growing on an image starting with each pixel as a separate segment. As a first step, neighboring pixels which lie at zero distance in the measurement space, thus have exactly the same pixel values in all channels, are joined into one segment. This first step in the process is is called zero distance merge (ZDM[1]). The remaining segments are considered during the best merge process.

In the best merge algorithm a merge cost function and stopping criterium are defined. Examples of merge cost functions are given in table 4.1. The merge cost function calculates the 'cost' of merging two adjacent segments in the image. The merge cost function that will be used is the Schoenmakers cost function. It calculates the euclidean distance, in the measurement space, between two neighboring segments. In the calculations, the average of all pixels in a segment is used.

Name	Cost function	Origin
Schoen	$\sum_{c=1}^{n}(v_c^i - v_c^j)^2$	as used in [Sch95]
B&G	$\frac{N_i * N_j}{N_i + N_j} * \sum_{c=1}^{n}(v_c^i - v_c^j)^2$	as used in [BG89]
SemiT	$4 * \frac{N_i * N_j}{(N_i + N_j)^2} * \sum_{c=1}^{n}(v_c^i - v_c^j)^2$	derived from [Til89]

with n the number of channels, v_c^k the intensity value of channel c of segment k (the average value of all the pixels in segment k) and N_k the number of pixels in segment k.

Table 4.1: Different merge cost functions

In each step of the best merge process, the lowest merge cost for 2 adjacent segments is calculated. The pair of segments having the lowest merge cost will be merged. The process is stopped when the merge cost becomes to high and

[1]ZDM will be used as an abbreviation for zero distance merge

the stopping criterium is met. The stopping criterion is usually an upper limit for the merge cost. For a detailed description of the best merge algorithm see [BG89, Sch95].

During each iteration, the best merge algorithm merges pairs of segments having lowest merge cost. This can reduce the total number of iterations necessary to compute the segmentation compared to Beaulieu & Goldberg's HSWO algorithm where only one pair of segments is joined each iteration. A reduction will only occur if there are more pairs of neighboring segments with lowest merge cost in a certain iteration. This will depend on the data of the input image and chosen cost functions and thresholds. It is not possible to gain a speedup by in some way manipulating this number of merged segments in a general case.

4.2 The role of preprocessing

At the start of the best merge process each pixel in the image belongs to a separate segment. The number of segments is equal to the number of pixels in the image. The run time of best merge is directly related to the number of segments in the image. A way to decrease the run time of best merge is to decrease the number of initial segments in the image as is done during ZDM. Normally, the number of segments joined during ZDM is small. This is caused by the fact that values of adjacent pixels hardly ever match in every channel at the same time. The chance that this occurs is very slim.

The image suba.lan (figure A.4) is a 256 x 256 area taken from the larger satellite image subjune.lan which can be seen in figure 4.1. The suba image contains 65536 (256x256) pixels, initially all assigned to separate segments. The number of initial segments remaining after ZDM is 65494, meaning that zero distance merge eliminated 42 segments. This is only 0.064%, a very small percentage. Increasing the number of elliminated segments at the ZDM stage i.e. decreasing the number of initial segments will leave less work for best merge. Fewer iterations are necessary to reach a segmentation result and therefore a speed up of the segmentation process is achieved.

There are two main approaches to reduce the number of initial segments in the image:

- Do not demand an exact match of neighboring pixels, but allow some deviation bounded by a threshold or extra parameter.
- preprocess the image by giving pixels which belong to the same field or ground cover class the same gray values. Then feed the modified image to best merge. In this way best merge is "fooled" into thinking that these pixels have spectral distance zero and will merge them during zero distance merge.

Reducing the number of segments as in the first sense is not yet investigated. The focus lies on the preprocessing of the data in the image space by clustering pixels by their spectral properties and modifying the image data to reflect this similarity. This approach is discussed in the the rest of this chapter.

As described in chapter 2, clustering of pixels is done by using the information which is present in the data. This can be done i.e. by recognizing large contiguous areas in the image. The suba image contains large areas with very distinct properties. In the image two streams can be seen as well as numerous large agricultural fields. Pixels in one of these 'fields' in the image all have similar spectral values within certain bounds and will therefore form groups or clusters in the image's measurement space. By finding these clusters in the image's measurement space, pixels in the image that are clustered together can be given the same spectral values in all

Figure 4.1: The subjune image; a 1953 x 1801 Landsat image in 6 spectral bands; from subjune.lan ©Eurimage (reduced in size)

channels. These values can simply be the average of the pixels within the cluster. The best merge algorithm can then, during zero distance merge, join the adjacent pixels in the cluster into a segment. In this way the number of initial segments for best merge can be reduced.

In order to find these groups in the measurement space a clustering algorithm can be used. There are several well known clustering algorithms like K-means and ISODATA [For73]. These will partition the pixels in the image in a predetermined number of groups or clusters. There are also algorithms for finding pixels which lie in one specific cluster. This can be useful when an image contains one large contiguous area, which is the case in the subjune image.

Treating the large water area in the subjune image, figure 4.1 and A.2, as one initial segment can eliminate approximately 20% of the pixels. The subjune image is a 1801 x 1953 lanfile in 6 spectral bands, containing 3517353 pixels. Eliminating 20% of these pixels reduces the number of pixels by 7034701. The number of iterations is reduced by 20% as well as the number of clusters which reduces greatly the number of calculations for finding the best merge pair during an iteration. Reducing the number of segments reduces the search time for the most similar pair of segments, reducing the time necessary to compute the segmentation.

The clustering algorithm used in the preprocessor is a form of nearest centroid sorting, a variant on the K-means algorithm. The implementation of the clustering algorithm was discussed in chapter 3.

4.3 Objectives of using preprocessing

The following objectives can be named:

- Decreasing the run time of the total segmentation process.
- Recognizing large areas in the image so that they are not split up during best merge and can be processed as one segment.

Using a preprocessor as a step in the segmentation to reduce the run time of the process makes it necessary that the preprocessing is not very time consuming. It is evident that the total run time using the preprocessor should be lower than without using the preprocessor. Each preprocessng step has to be implemented by an algorithm with low time complexity.

If an image contains a few large areas and a lot of much smaller ones, the large areas will often be left split up in separate segments by best merge. This is caused by the fact that the threshold on the merge cost has to be set such that not too many segments in the image are wrongly joined or split up. Choosing too high a threshold will give a lot of wrongly joined segments. Choosing too low a threshold gives a lot of wrongly split up segments. For a discussion on the effect of different theshold values on the final segmentation result see [Voo95, TES94].

In a large area in the image, the spectral variance for pixels which lie well separated, but belonging to the same segment, can be sufficiently large that this causes the merge cost of two segments in the area to exceed the set threshold value. Segments which should be merged are not. Usually, the threshold has to be chosen in such a way that merge costs of segments in the majority of the real fields in the image do not exceed this threshold value. If a large area in the image can be treated as an initial segment, the segmentation threshold for the image can be chosen lower preventing overgrowing of regions.

4.4 The preprocessing steps

The different preprocessing steps are shown in figure 4.2, references are given to diagrams explaining the different steps.

First the image is read and stored in memory. The second step is to cluster the image data. A diagram explaining the clustering algorithm used is shown in figure 3.1 Ways of clustering the image data were explained in chapter 3. The output generated by the clustering algorithm is a cluster map with in each cell the cluster number to which the corresponding pixel is assigned. Thresholding is then performed on the pixels in the different clusters. Pixels which differ more than (threshold x standard deviation) from their cluster centroids are marked as unclustered in the cluster map. The next step is to create a new image in memory based on the information in the cluster map. Depending on the chosen output format, cluster marking or area marking is used to create the new image.

- Using cluster marking, average values are substituted for the pixels in the same cluster.
- Using area marking, the average values of each area in a certain cluster (figure 4.4(b)) are substituted. Area marking is described in more detail in section 4.5

The last step in preprocessing is to write the image which has been built in memory to a file in ERDAS-74 format. This output image can be used as input for the best merge algorithm.

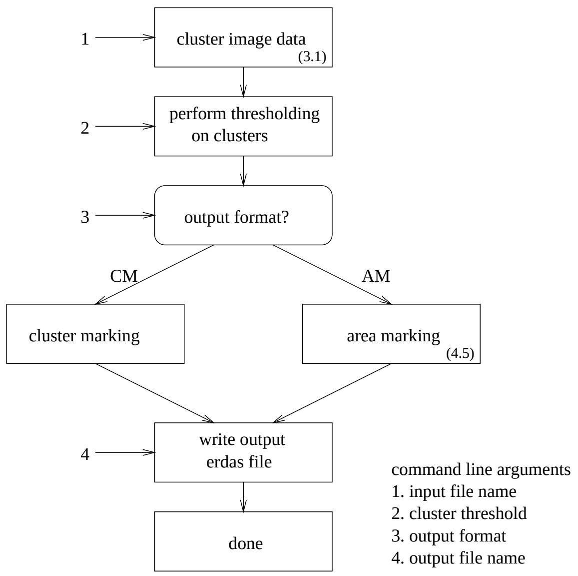


Figure 4.2: The subsequent preprocessing steps

4.5 Area finding algorithm

In this section the design of the area finding algorithm is discussed. First the need for an area finding algorithm as part of the preprocessor is discussed. Then the design of the algorithm is given.

4.5.1 Need for an area finding algorithm

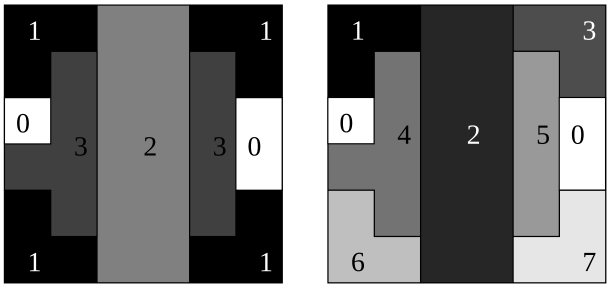
The clustering algorithm divides the input image into clusters 1 to n, where n is the number of seed points provided. Cluster 0 contains the pixels for which the cluster threshold was exceeded (these pixels were left unclustered). Figure 4.3 shows a clustering result showing only one cluster. The image which was clustered is suba.lan, part of the subjune image (figure 4.1). The white pixels belong to the cluster. The other pixels are depicted at half intensity.

Figure 4.3: suba.lan, a cluster in the image

In this case, there are two large *separate* areas in the image whose pixels belong to the same cluster. In the cluster marking approcach, all pixels in a cluster are averaged and this average is substituted for these pixels in a new image file. This can cause the cluster average to differ much from the averages of the pixels in the different areas in the image (which can be spread over large distances in the image space). This has the unwelcome effect that cluster averages (when substituted) can differ a lot from non clustered pixels belonging to the same region (field). This increases the merge cost for these pairs of segments during best merge, causing regions to be left split while they should have been joined by best merge in the segmentation process.

To overcome this problem, the pixels in each cluster will be partitioned according to the area in the image (image space) to which they belong. For each area belonging to a cluster, a separate average based only on pixels in this bounded area is calculated and substituted in a new image file. In order to partition each cluster into areas, an area finding algorithm has to be developed.

The working of the area finding algorithm will now be illustrated by a simple example. After clustering, a cluster map of the image is available. This is shown in figure 4.4(a). Each cell of the map corresponds to a pixel in the image and contains the cluster number of the pixel.



(a) cluster map (b) resulting area map

Figure 4.4: The result of area marking on a cluster map

Three clusters have been found. Note that in this example there are a few unclustered pixels (marked with 0). Pixels in the 3 clusters fall into 7 different areas in the image. Figure 4.4(b) shows the 7 areas. The area finding algorithm has to be able to locate these areas and process them separately.

4.5.2 Design of the area finding algorithm

Input for the area finding algorithm is the two dimensional pixel cluster map as produced by the clustering algorithm after cluster thresholding is performed. The number in cell (y,x) indicates the number of the cluster to which pixel (y,x) is assigned. The area finder locates areas which are groups of connected pixels having the same cluster number in the cluster map. In the pixel area map, these areas are marked with a unique number. The output of the area finding algorithm is a 2-dimensional area map of the image with in each cell the number of the area to which the corresponding pixel belongs. This map can be used to create an output file in which all pixels in the image file having the same area number are given the average pixel value of the pixels in that area.

The basic processing steps of the area finding algorithm are shown in figure 4.5. The flow diagram of the algorithm uses pseudo C code. "=> (r,e)" indicates that

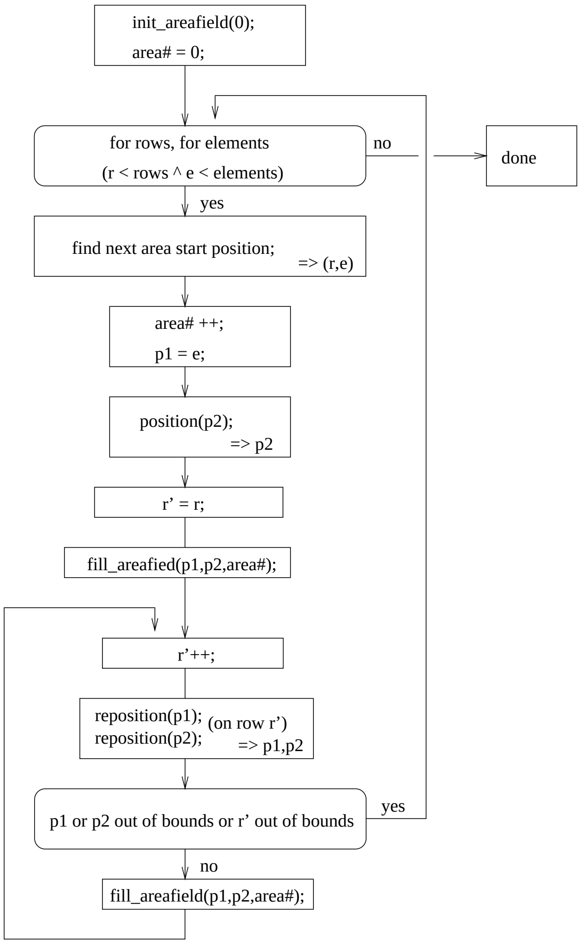


Figure 4.5: Basic processing steps of the area finding algorithm

a value for r and e is retruned by a subroutine. In the diagram r and e indicate the actual row and column positions at a certain point during execution. r' is used in the actual search for area boundaries for each separate area in the cluster map. p1 and p2 are used to mark the start and end pixel position of the current area on the current row (r or r'). The subroutines used will now be described:

- The area map is initialized with 0 in each cell.

- **find next area start position**: A new area may be started if the following conditions are all satisfied:

    1. a cluster number change occured going from pixel i-1 to pixel i on the current row of the cluster map;
    2. pixel i on the current row in the cluster map has a clusternumber > 0 (is a clustered pixel).
    3. pixel i on the current row does not yet have an area number (area numbers are > 0).

  Returned value is (r,e), the next (row, column) position in the cluster map that satisfies these conditions.

- **position p2**: p2 is positioned to the left of the position where the next cluster number change in the cluster map occurs. The first row of pixels in the new area is now marked with the pointers p1 and p2.
- **fill_areafield(p1,p2,area#)**: the current row in the area map is filled with the value area# from position p1 to position p2.
- **reposition(p1)**: After moving down one row, p1 is moved to the left or right respectively to point to the left boundary of the cluster on the new row. p2 is repositioned in the same way to the right boundary. Figure 4.6 shows this repositioning.

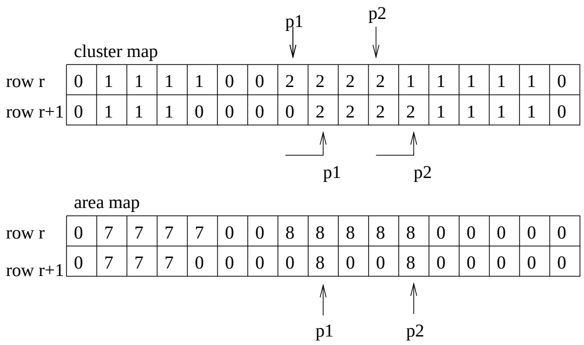


Figure 4.6: Repositioning of area marker pointers

The area finding algorithm will assign an area number to each pixel in the image. Pixels that were left unclustered by the clustering algorithm will be assigned area number -1. This means that these pixels can be treated separately in further calculations.

4.6 Effect of preprocessing on the segmentation result

Preprocessing tries to give large regions of pixels in an image the same average gray values. It is to be expected that this has an influence on the segmentation result.

- The quality of the resulting segmentation will be lower than if preprocessing were not used.
- There will be a significant run time improvement of the best merge process using preprocessing.
- Edge detection can probably not be used as combined preprocessor because parts of fields may be found by classification and given the same (average) color. This may differ enough from the rest of the field to cause the edge detector to find a nonexistent edge (which would not be found without preprocessing).
- Using cluster marking will cause the resulting segmentation to be of lower quality than using area marking.
- increasing the cluster threshold of the preprocessor will cause a decrease in segmentation quality.

In section 5.3 the preprocessor will be tested showing the influence preprocessing has on end result of the segmentation.

Chapter 5

Preprocessor testing

In this chapter the preprocessor will be put to the test. An evaluation of the quality of the segmentation results will be given relative to quality achieved by using best merge stand alone. The computation times will also be compared.

5.1 Measurement of segmentation quality

In order to give a quantative measure for the quality of the segmentation of an image, this segmentation can be compared to a "true" segmentation. Differences between the "test" and "true" segmentation can be expressed by a quality value. In order to find this value, a "true" segmentation of the image must be known. Obtaining a "true" segmentation of an image can be done by "going into the field" and measuring every feature in the terrain. This information can be "mapped" onto the satellite image, giving a "true" segmentation.

A more practical way of obtaining a "true" segmentation is described in [TES94]. It is based on a method for generating artificial satellite images, constructed based a man-made drawing. Each segment in the image is filled with the texture pattern of a chosen ground cover class, according to ground truth data of a given set of satellite images. Because all the information in the artificial image is known, a "true" segmentation can be determined.

When segmenting an image, two types of errors can be made causing a decrease in segmentation quality when compared to the "true" segmentation:

- split errors: a real field is split into one or more segments
- merge errors: pixels of different fields are merged into one segment

These two types of errors are used to quantify the quality of the segmentation result and give the split quality and merge quality. "Test" segmentation will denote a segmentation result of an artificial image obtained using a certain segmentation method.

For each field i in the "true" segmentation the number of pixels, nin_i is calculated. For each segment j in the "test" segmentation, the number of pixels $ncov_{ij}$ from field i in the "true" segmentation is determined. In [TES94] two split quality measures for a field i are defined. The coverage quality measure tries to find the segment j with the largest coverage $ncov_{ij}$ and calculates:

$$Qsplitcov_i = \frac{ncov_{ij}}{nin_i}$$

This measure is 1 if the field i is contained in exactly one segment in the "test" segmentation. If field i is split into more than one segment in the "test" segmentation the split quality will be less than one. In the extreme case, $Qsplitcov_i$ aproaches $\frac{1}{nin_i}$.

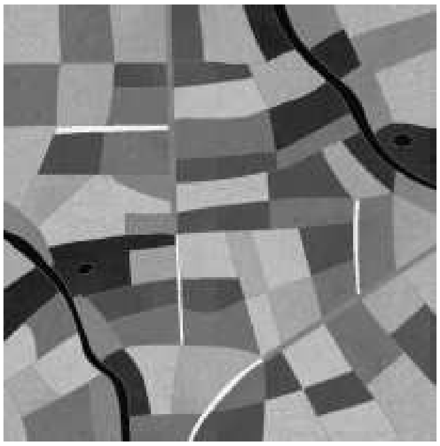
The other quality measure which can be used is the split entropy:

$$Qsplitent_i = -\sum_j p_{ij} * log(p_{ij}) \; with \; p_{ij} = \frac{ncov_{ij}}{nin_i}$$

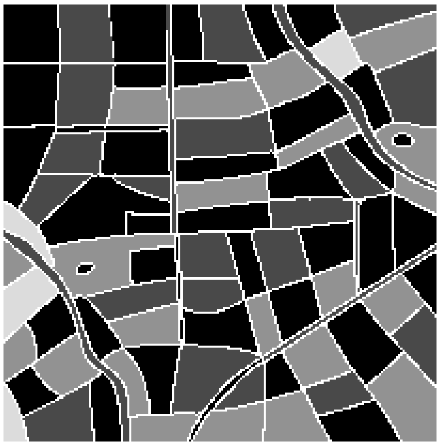
The entropy measure is 0 if the field i in the "true" segmentation is not split in the "test" segmentation and is large when the field i is heavily split (approaching $log(nin_i)$). This means that the segmentation quality is high when $Qsplitent_i$ is low.

Merge qualities are defined by switching the roles of the "true" and "test" segmentation. Quality measures for an entire image are calculated as the average of the qualities per field weighted by the number of pixels in the field.

Figure 5.1(a) shows an example of an artificially generated image together with a "true" segmentation as described in [TES94].



(a) Channel 5 of an artificially generated Landsat image, a.k.a. z.lan.



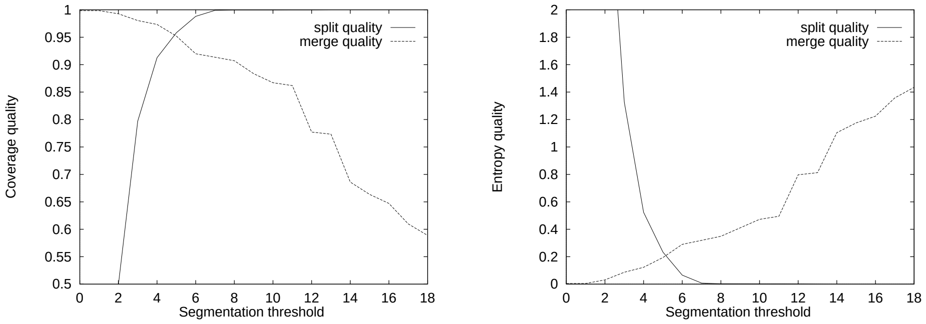
(b) "True" segmentation of 5.1(a)

Figure 5.1: Segmentation of an artificial satellite image

Using the best merge algorithm "test" segmentations using thresholds from 1 to 18 have been made of 5.1(a). The cost function used is Schoenmakers. In figure 5.2(a) and 5.2(b) coverage and entropy quality is shown for the segmentations at different thresholds.

When looking at the coverage measure, split quality increases together with the segmentation threshold. The opposite happens with the merge quality. This is caused by the fact that best merge starts with each pixel in a separate segment and repeatedly merges the pair of segments with lowest merge cost. The process stops when the lowest merge cost exceeds the segmentation threshold. At low thresholds the merge cost quickly exceeds the threshold value leaving leaving a lot of small segments. This means that a lot of fields are left split up while they should have been merged, leading to a low split quality. This also means that not many fields have been merged unjustly, leading to a high merge quality. At high thresholds, the opposite happens, few fields are unjustly split while more fields are unjustly merged.

When using the entropy measure, figure 5.2(b), segmentation quality is high when entropy is low, giving a behavior which is opposite to the coverage quality measure.



(a) Quality based on coverage

(b) Quality based on entropy

Figure 5.2: Quality measures as function of the segmentation threshold, 'schoenmakers' cost function

For a given segmentation algorithm, the split and merge quality can be seen as a function of the parameters of the method. The best merge algorithm has one parameter: the segmentation threshold. The optimal segmentation threshold can be chosen such that split and merge quality are equal. At this threshold the worse of the 2 qualities is the best. In this case the optimum threshold lies at 5. This can be seen in both the coverage and entropy graph.

5.2 Segmentation quality and segmentation speed

The objective of this research project is to develop a method to speed up the best merge algorithm compromizing as little as possible the quality of the segmentation result. This means that the quality of the result is a combination of segmentation quality and segmentation speed. As a reference, the cpu time of the best merge algorithm on the image of 5.1(a) will be used. The cpu time is defined as user time + system time. It is clear that a higher threshold dramatically increases the necessary cpu time for best merge. This is caused by the fact that at higher thresholds more merge steps are carried out than at low thresholds, increasing the cpu time. This is shown in figure 5.3.

The cpu time has been measured on a sun 4/670 at low load conditions. All tests described in the rest of this chapter will be carried out on the same machine under similar conditions, making the timing results comparable. The cpu time shown in the diagrams is the total cpu time including cpu time used for preprocessing. The cpu time for preprocessing will also be given and is (of course) independent of the segmentation threshold used for best merge.

5.3 Test cases

The preprocessor as implemented can be configured using several parameters. The 3 most important parameters are:

- seed point selection (from file or automated)
- clustering threshold

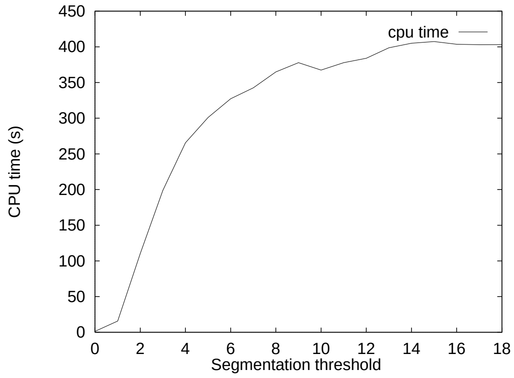


Figure 5.3: CPU time necessary for best merge on 5.1(a); no preprocessing.

- output format (cluster marking or area marking)

In the following test cases the effect of these parameters on the resulting segmentation is studied. In each case preprocessing is done followed by best merge.

Quality of the results is indicated by split and merge quality and will be depicted in graphs. Split quality gives an indication of the number of "true" fields unjustly split into more segments. Low split quality indicates that a high number of "true" fields were split up. Merge quality gives an indication of the number of "true" fields unjustly merged into one segment. Low merge quality indicates that a high number of "true" fields were unjustly merged.

In the test cases, only the coverage quality graphs will be given. The Entropy graphs do not contribute any more valuable information about the segmentation quality.

5.3.1 Effect of preprocessor's output format on the result

This test case illustrates the effect of the chosen output format on the resulting segmentation. This means that the output format is the only varying preprocessor parameter in the experiment. First cluster marking is chosenas ouput format. The parameters for the preprocessor are as follows:

- seed points from file "ground-truth-portugal", see table 3.1.
- clustering threshold set at 1.0 x standard deviation.
- input file "z.lan" (see figure 5.1(a)).

Figure 5.4 shows the effect on the quality of the resulting segmentation using cluster marking, labeled with "split quality pp" and "merge quality pp". To be able to compare the results, the segmentation quality using best merge without preprocessing is also plotted (labeled with "split quality bmerge" and "merge quality bmerge").

The quality of the segmentations using cluster marking is slightly lower than the best merge result over almost the whole threshold range. Cluster marking can cause clusters to be formed of pixels which should lie in different "true" segments. This in turn causes the zero distance merge (ZDM) [1] to form initial segments containing pixels from more than one "true" segment. These types of errors cannot be corrected

[1]ZDM is used as an abbreviation for zero distance merge, the first phase of the best merge process where pixels at spectral distanze zero are merged in to larger segments. Chapter 4 gives more information

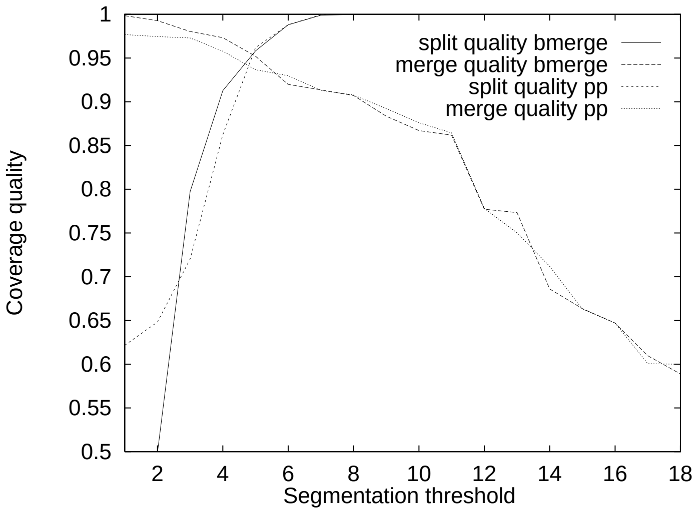


Figure 5.4: Quality of segmentation using preprocessing with cluster marking

by best merge and will therefore decrease the merge quality of the segmentation. Using preprocessing, merge quality will hardly ever be perfect.

Another effect of cluster marking is that below threshold 2 the split quality is considerably higher than the best merge split quality. This is because ZDM merges a lot of initial segments. as found during clustering. The number of segments after ZDM is 15183. Without cluster marking best merge reports 35509 segments after ZDM, causing the split quality to be a lot lower. As a reference, the total number of pixels in z.lan is 41616. These figures show an unusually high reduction in segments after ZDM when preprocessing is not used. Normally this reduction is negligible, see chapter 4.

Low thresholds (between 2 and 5) show at least a 10 % sqlit quality decrease. This splitting up of segments is caused by the fact that cluster marking is used, it causes the merge cost threshold to be exceeded when adjacent unclustered and clustered pixels are considered for best merge. This is due to the fact that with cluster marking, pixels spread over the whole image belonging to the same cluster are averaged, increasing the variance with adjacent non clustered pixels. Using area marking instead of cluster marking reduces this effect. This is shown in figure 5.5. The input file and cluster threshold have been chosen the same as with cluster marking.

The effect of decrease in split quality at low thresholds is elliminated using area marking. Above threshold 2 the curve is practically indentical to the best merge split quality curve. Therefore using area marking gives a slight quality improvement over cluster marking. For an illustration of this effect of cluster marking see table 5.3.

Figure 5.6 shows the effect on the cpu time using area marking. The cpu times in the test case using cluster marking and area marking include the preprocessing time. Preprocessing using cluster marking gave comparable timing results.

Both preprocessing using cluster marking and area marking show a sharp decrease in cpu time (above threshold 1 compared to best merge stand alone. At threshold 5, the segmentation threshold giving optimal segmentation quality, the cpu time goes from 300 to 100 seconds, a decrease of 200 seconds or 66%. The effect on the cpu time will be investigated further in another test case using larger images.

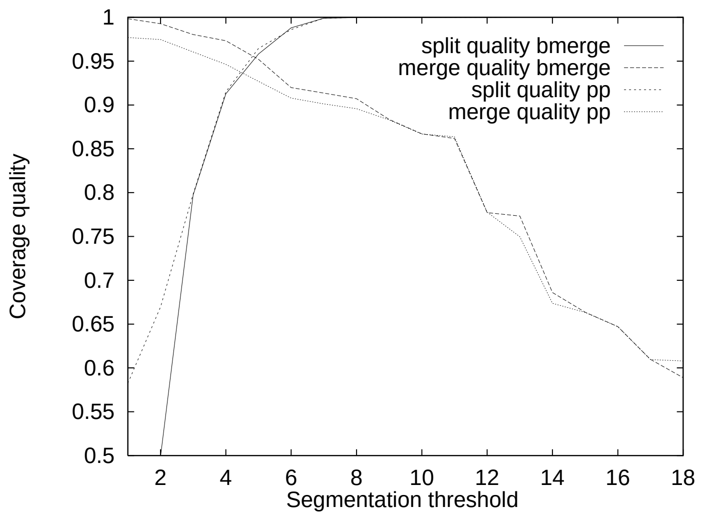


Figure 5.5: Quality of segmentation using preprocessing with area marking

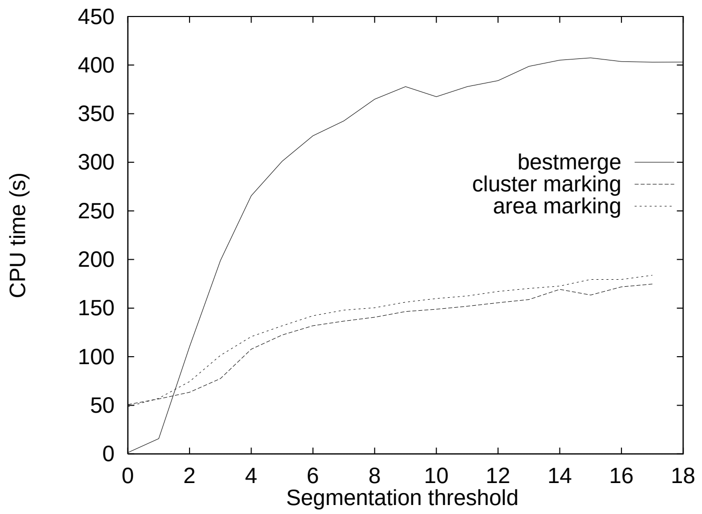


Figure 5.6: CPU time necessary for best merge on 5.1(a); time for preprocessing: 46.5 s for cluster marking, 45.5 s for area marking.

5.3.2 Effect of seedpoint selection on the result

Instead of using the Portugal seed points, seed points in this test case are found using the **find_seedpoints** routine described in section 3.1.1. 5 seed points which are sufficiently dissimilar were found (see section 3.1.2). During the test area marking is used because it gave superior quality to cluster marking in the previous test case. Other parameters are kept constant to be able to compare the results. The input file is "z.lan". In figure 5.7 the results using automated seed point selection using area marking are shown.

Automated selection of seed points produces a segmentation of lower quality compared to using the Portugal seed points. This is mainly due to the decrease in merge quality when using automated seed point selection. If automated selection places a seed point close to the border between two "true" segments in the image with the same ground cover type, these two segments may be joined during

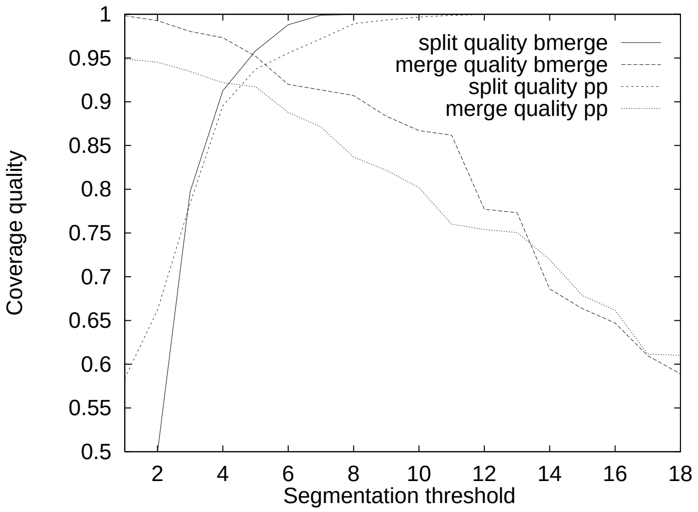


Figure 5.7: Quality of segmentation using preprocessing with automated seed point selection using area marking

clustering. This can not be corrected by the area marker causing a decrease in merge quality. A decrease in split quality is less likely because clustering tries to merge similar pixels rejecting pixels which are too dissimilar (more dissimilar than the cluster threshold. Note that cluster thresholding is only applied to the final clustering result).

To increase the quality of the segmentation result using automated seed point selection, a better seed point selection algorithm could be developed. An other way of increasing the quality of the result is to use a lower clustering threshold. The effect of the cluster threshold setting is investigated in the next test case.

Table 5.1 shows the optimal best merge thresholds for the different preprocessing strategies. The results show that the quality of the resulting segmentation is lower than when using best merge stand alone. The segmentation thresholds shift slightly compared to the optimal best merge threshold of 5.0. Automated seed point selection gives the worst quality, however still above 90%. Table 5.2 and 5.3

	Crossover point				*Total CPU time (s)*	
Preprocessing	*coverage*		*entropy*		*pre-*	
method	*thresh.*	*quality*	*thresh.*	*quality*	*processing*	*bmerge*
none	5.0	0.954	5.0	0.214	0.0	301
Cluster marking	4.7	0.938	5.1	0.232	46.5	68
Area marking	4.4	0.937	4.8	0.251	44.5	64
A.M. rand. seeds[2]	4.5	0.918	5.1	0.351	65.5	67

Table 5.1: Effect of the output format on the quality

give an indication of what caused the loss of quality in the different segmentations. The "true" segmentation contains 93 regions of which 86 are regions with interior pixels, regions of size ≥ 5 pixels. Best merge at threshold 5 found 98 real regions and 2488 regions in total. This means that there are 2390 regions with 1 to 5 pixels. For the different preprocessing methods, table 5.1, these values are also calculated.

[2]Random seed point selection was used together with area marking

In each case the cluster threshold is 1.0 x standard deviation (as used in the test cases). The best merge threshold has been set to achieve optimal coverage quality (see table 5.1).

Segmentation	# of regions	with inner pixels
true	93	86
bmerge (thresh. 5)	2488	98
cluster marking[3]	2673	119
area marking[4]	2797	106
random seeds[5]	2467	119

Table 5.2: Size of the regions found in the different segmentations

The results in table 5.2 explain that the merge quality in all three preprocessing methods is high at the crossover point because there are not drastically more large regions (with inner pixels) than in the best merge result. The same holds for the split quality: using the different preprocessing methods produces roughly the same number of regions (usually a slightly higher number).

preprocessing method	bmerge thresh.	segments after ZDM	segments after best merge	# isol. pixels	# of pixels in largest region
none	4.0	35509	3661	2248	968
	5.0		2823	1853	999
	6.0		2176	1530	1208
Cluster marking	4.0	15183	4109	2348	1132
	4.7		3119	1927	1172
	5.0		2810	1819	1177
	5.1		2731	1781	1178
	6.0		2195	1528	1211
Area marking	4.0	15423	3670	2189	1141
	4.4		3276	2005	1152
	4.8		2959	1880	1174
	5.0		2806	1812	1177
	6.0		2168	1511	1210
Random seed selection & area marking	4.0	14243	3301	1904	1203
	4.5		2915	1724	2467
	5.0		2643	1622	2285
	5.1		2575	1598	1239
	6.0		2130	1389	1245

Table 5.3: Best merge statisics around optimum segmentation threshold

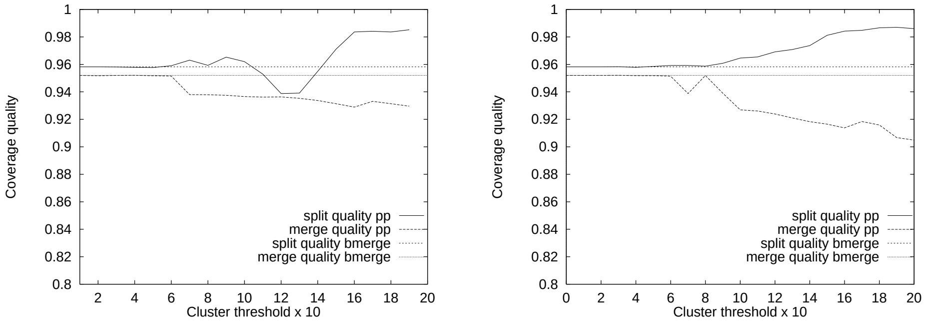
The fact that quality of the segmentation results using preprocessing is almost equal to the best merge quality also concurs with the figures in table 5.3. An observation that can be made is that cluster marking produces far more isolated pixels than area marking does. This is an illustration of the assumption made in section 4.5, the reason why area marking was introduced as part of the preprocessor.

[3] clustering at threshold 1.0; best merge at threshold 4.7
[4] clustering at threshold 1.0; best merge at threshold 4.4
[5] clustering at threshold 1.0; best merge at threshold 4.5

5.3.3 Effect of cluster threshold on the result

Until now, the cluster threshold has been kept constant at 1.0 x standard deviation. In this section, the effect of varying the cluster threshold will be investigated. For this test, the input image will be "z.lan". The best merge threshold will be kept constant at 5, the optimum threshold as found in other experiments. The preprocessor's cluster threshold will be varyied from 0.1 x standard deviation to 2.0 x standard deviation.

Figure 5.8(a) and 5.8(b) show the effect on the coverage quality using cluster marking and area marking. These graphs show that beyond a cluster threshold of



(a) using cluster marking

(b) using area marking

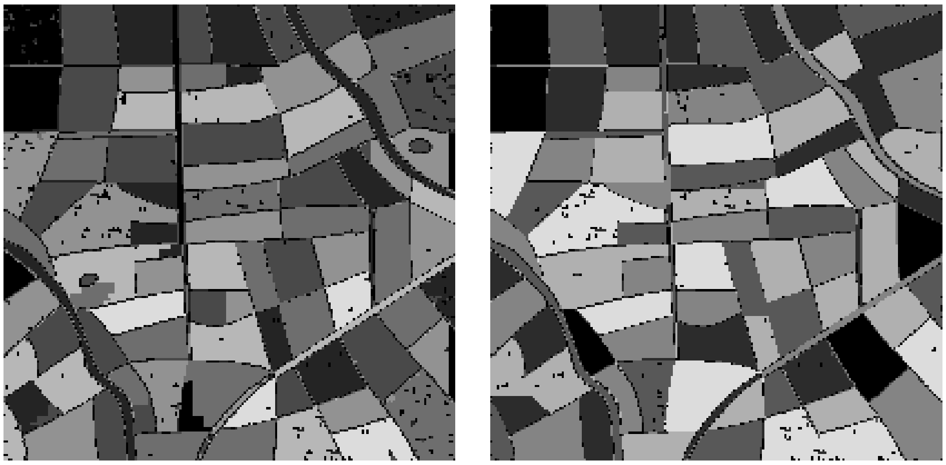
Figure 5.8: Effect on segmentation quality of varying cluster threshold

0.8 or 0.9, the the merge and split quality of the segmentations using preprocessing deviate from the quality levels set by best merge. In [TES94] it is stated that an overall quality value for a segmentation may be calculated by taking the average of the split and merge quality. In the case of area marking, figure 5.8(b), this would mean that the overall quality would approximately average out to the best merge overall quality over the whole cluster threshold range.

Figure 5.9(b) shows that overall quality using preprocessing cannot be taken as the average of merge and split quality at high preprocessing cluster thresholds. The result can be compared to the "true" segmentation shown in figure 5.1(b) and the stand alone best merge result in figure 5.9(a). In this case, clustering has been done at threshold 2.0 using area marking. In this segmentation, the segments joined incorrectly mainly belong to fields in the image of the same ground cover class, separated by weak boundaries. A poor segmentation result does give a high segmentation speed. In this case just 50 seconds including preprocessing. A segmentation result at cluster threshold 0.6 is shown in figure 5.10(a). This segmentation result is much better compared to the one at cluster threshold 20 however, the cpu time necessary to compute the segmentation result is 200 seconds, not much lower than if best merge were used stand alone.

Figure 5.10(b) shows the segmentation result using cluster threshold 5, the optimal threshold as deduced from figure 5.2(a).

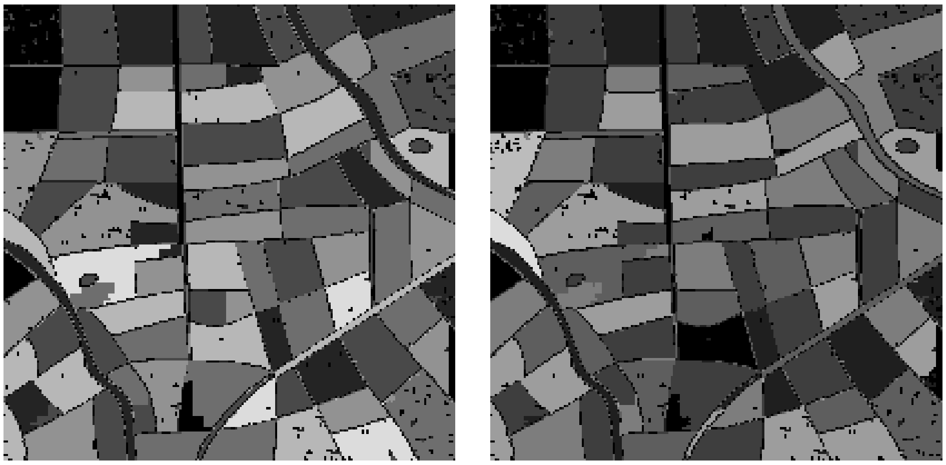
Varying the preprocessor's cluster threshold also influences the number of segments best merge finds during ZDM and the cpu time for best merge. Figure 5.11(a) shows that with increasing preprocessor threshold, both the number of segments after ZDM decreases and the cpu time decrease. The influence is shown for both cluster marking and area marking, with little difference between the two. Differences in the number of segments are in the order of 10 to 100 (figure 5.11(a)) which is not shown in this scale. Cpu time for preprocessing has been set at 42



(a) Best merge at threshold 5

(b) Clustering at threshold 2.0 with area marking, best merge at threshold 5

Figure 5.9: Effect of high preprocessing cluster threshold on the segmentation result



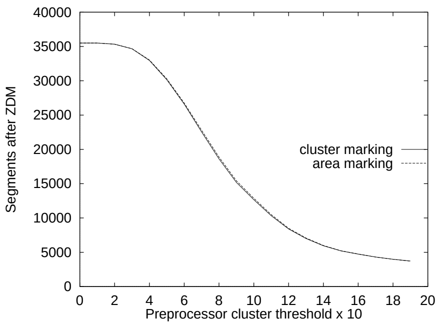
(a) Clustering at threshold 0.6 with area marking

(b) Clustering at threshold 1.0 with area marking

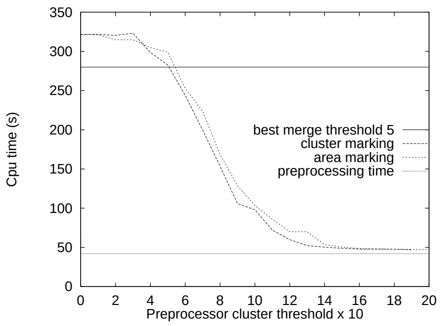
Figure 5.10: Clustering at different thresholds followed by best merge at threshold 5

seconds (the worst case). Over the whole cluster threshold range there was only a difference of $\pm$ 0.5 seconds. This gives a minimum reference for the cpu time. Cpu time using best merge without preprocessing at threshold 5 takes 280 seconds and is given as a maximum reference.

Except for very low cluster thresholds there is an improvement on the cpu time as opposed to using best merge stand alone. An optimal configuration minimizes cpu time while maximizing the quality of the result at a given segmentation threshold. For the segmentation threshold set at 5, the highest quality that can be reached using preprocessing is 0.954, as can be seen in table 5.1. Usin g area marking, the cluster threshold should not be set far beyond 1.0 because then the quality curves



(a) Effect on number of segments after ZDM



(b) Effect on cpu time

Figure 5.11: Effect of varying cluster threshold using fixed best merge threshold (5)

with preprocessing start to deviate from the best merge quality level of 0.954.

5.3.4 Tests using the hybrid segmentation method

In this section the preprocessor is tested together with Schoenmakers hybrid segmentation method (section 1.4.1). It uses edge detection and edge joining to find closed polygons in the image which can be processed independently (possibly in parallel, speeding up the segmentation process. The minimum polygon size and best merge threshold can be set by the user. Regions are not allowed to grow outside the borders of a polygon.

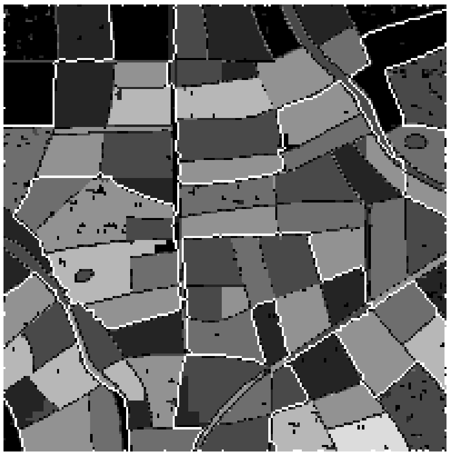
Figure 5.12(a) shows the segmentation result on z.lan using a minimum polygon size of 6 and best merge threshold 5. The polygons are the white contours. Figure 5.12(b) shows the segmentation result when preprocessing is used. Preprocessing causes a lot of edges to be found giving a large number of small polygons. This

(a) without preprocessing

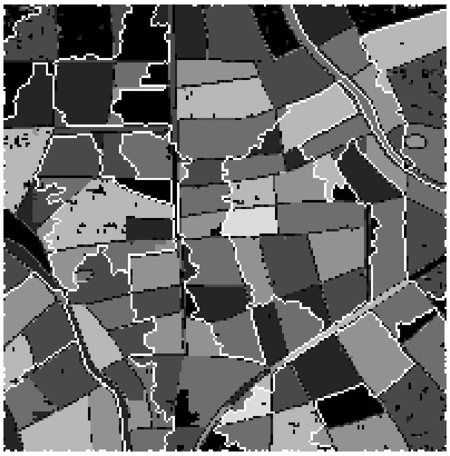
(b) using area marking at threshold 1.0

Figure 5.12: Hybrid segmentation with minimum polygon size 6 and best merge threshold 5

problem can be overcome by choosing a larger minimum polygon size. The results are shown in figure 5.13(a) and 5.13(b). In this example a polygon size of 1000 was chosen. The result using preprocessing is much better using large polygons. Table 5.4 shows that using the preprocessor with small polygon size does not give



(a) without preprocessing



(b) using area marking at threshold 1.0

Figure 5.13: Hybrid segmentation with minimum polygon size 1000 and best merge threshold 5

a decrease in cpu time. The resulting segmentation is also of poor quality due to the large number of small segments formed due to the small polygons. Using a large polygon size gives acceptable run time performance (it was not slower). The segmentation result shows that polygons were in some cases drawn in the wrong locations causing fields to be left split up during best merge. The segmentation

Polygon	no preprocessing		preprocessing	
size	cpu time	# of polygons	cpu time	# of polygons
6	0 + 72	150	45 + 48	895
1000	0 + 115	14	45 + 41	17

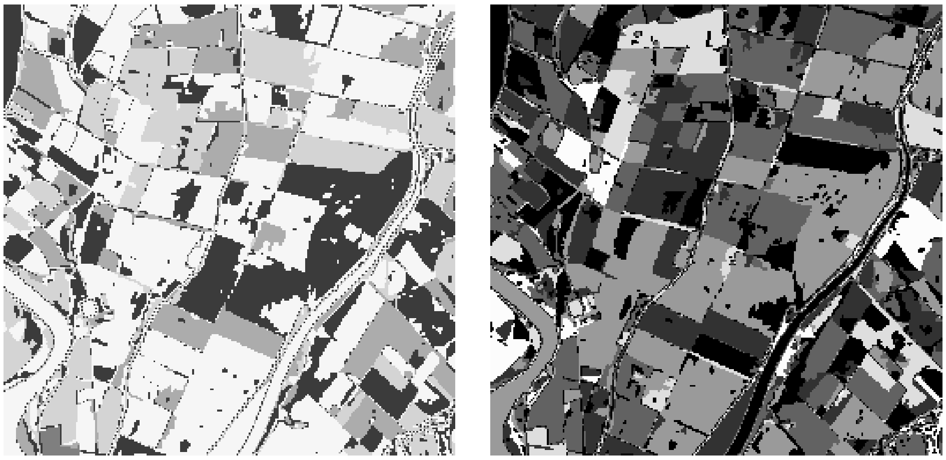
Table 5.4: Hybrid segmentation statistics

results shown in this section can be compared to the segmentation result in figure 5.9(a) where best merge at threshold 5 was done without preprocessing and figure 5.10(b) showing a segmentation result using area marking at threshold 1.0.

5.3.5 Tests on other images

The preprocessor has also been tested on other images where no "true" segmentation is known. Cluster threshold has been set at 1.0 x standard deviation. Area marking is used. Best merge threshold is set at 7. Figure 5.14(a) shows the best merge segmentation result on the suba image using best merge threshold 7. This image is part of the portugal images, therefore the portugal ground truth classes are used as seed points.

Tables 5.5 and 5.6 show the differences between the segmentation of the suba image with and without preprocessing. Using preprocessing gives a 75% run time



(a) Best merge at threshold 7 (b) Clustering followed by best merge

Figure 5.14: Segmentation of the Suba image

improvement. This is due to the reduction of the number of segments after ZDM by a factor 2 when using preprocessing.

total # of pixels	65536
bmerge thresh.	7
cpu time	897 sec.
segments after ZDM	65494
segments after bmerge	3842
# isol pixels	2335
largest region	4867
# of regions	3622
# of regions > 1 pixel	1265
# of regions with int. pixels	307

Table 5.5: Best merge results of suba.lan without preprocessing

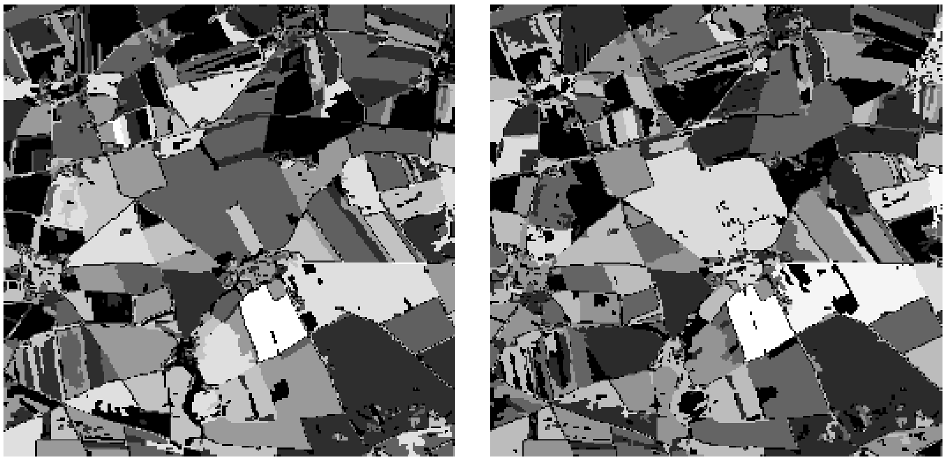
Using preprocessing gives slightly more regions than without using preprocessing. A conclusion that can be drawn based on the figures in the two tables is that using preprocessing causes the segmentation threshold to be exceeded faster than without using preprocessing. This appears mainly to be the case with larger segments because there was no increase in the number of isolated pixels. These observations concur with a visual assessment of the segmentation results. Comparing figure 5.14(a) and figure 5.14(b) also shows that in several cases different segments have been joined or remained split up in each segmentation, producing different regions. This could not be deduced from the numerical information gathered about the segmentations.

The Bern image comes from an other set of images. The image can be found in appendix A. The portugal ground truth classes cannot be used here. Instead, automated seed point selection is used. In this case, the number of seed points was set at 6. The cluster threshold was set at 1.0 and area marking was used.

Calculating seed points causes preprocessing to take longer than when using seed

cluster threshold	1.0
output format	area marking
cpu time preprocessing	137 sec.
total # of pixels	65536
bmerge thresh.	7
bmerge cpu time	240 sec.
total cpu time	377 sec.
segments after ZDM	32694
segments after bmerge	4034
# isol pixels	2295
largest region	4801
# of regions	3745
# of regions > 1 pixel	1435
# of regions with inner pixels	342

Table 5.6: Best merge results of suba.lan with preprocessing



(a) Best merge at threshold 7 (b) Clustering followed by best merge

Figure 5.15: Segmentation of the Bern image

poins from a file. The time for preprocessing was 100 seconds. This was caused by a fast stabilization of the clustering, only 7 iterations were needed. Best merge on the result took another 244 seconds. This is 1/3 the time needed for using best merge stand alone, this takes 921.5 seconds at threshold 7. After ZDM, only 31830 segments remained, compared to 63897 without preprocessing. When comparing the cpu times for best merge with preprocessing and stand alone, about the same ratio is obtained as in the suba segmentation. This shows that the cpu time necessary to compute a best merge segmentation at a certain threshold is closely related to the number of initial segments in the data. This concurs with the fact that best merge is a form of hierarchical clustering.

Tables 5.7 and 5.8 give some numerical statistics about the different segmentations as was done for the suba image.

The segmentation results using preprocessing show the same syptoms as the suba segmentation. Some detail has been lost and some fields have been joined

total # of pixels	65536
bmerge thresh.	7
cpu time	921.5 sec.
segments after ZDM	63897
segments after bmerge	3554
# isol pixels	1742
largest region	4180
# of regions	3327
# of regions > 1 pixel	1568
# of regions with int. pixels	316

Table 5.7: Best merge results of bern.lan without preprocessing

cluster threshold	1.0
output format	area marking
cpu time preprocessing	100 sec.
total # of pixels	65536
bmerge thresh.	7
bmerge cpu time	243.5 sec.
total cpu time	343.5 sec.
segments after ZDM	31830
segments after bmerge	3527
# isol pixels	1691
largest region	3863
# of regions	1521
# of regions > 1 pixel	329
# of regions with inner pixels	342

Table 5.8: Best merge results of bern.lan with preprocessing

which remained split in the stand alone best merge segmentation and vice versa. This can be seen in figure 5.15(a) and 5.15(b).

5.3.6 Segmenting larger images

To show the effect of the preprocessor when processing larger images, a 512 X 512 rectangle has been taken from the subjune image. In appendix A channel 5 of the selected rectangle is shown. The image contains a large water area and large as well as small fields. Seed points for the clustering were chosen from the portugal ground truth data. The cluster threshold was set at 1.0 and area marking was used. Best merge was then used at threshold 7. Figure 5.17 shows the resulting segmentation. Table 5.9 and 5.10 give numerical information about the two segmentations. The cpu time improvement is more than 3-fold with more than half the pixels merged during ZDM in the preprocessing segmentation.

The largest region, the large water area in the image, is also found in the segmentation using preprocessing, and is increased in size by 10%. The other values found follow the pattern found in the suba and bern segmentations.

The segmentation result using preprocessing is shown in figure 5.17. The result without preprocessing is shown in figure 5.16

Figure 5.16: Part of the subjune image; segmented by best merge at threshold 7

total # of pixels	262144
bmerge thresh.	7
cpu time	15807 sec.
segments after ZDM	261117
segments after bmerge	28633
# isol pixels	15610
largest region	80646
# of regions	26395
# of regions > 1 pixel	10705
# of regions with int. pixels	1733

Table 5.9: Best merge results of subjunepart.lan without preprocessing

5.4 Concluding remarks

The results of the tests described in this chapter indicate that using the preprocessor as a step in the segmentation process can reduce the computation time needed to obtain a segmentation result. This reduction in computation time is a tradeoff for a decrease in segmentation quality. The test cases showed that if the preprocessor parameters are chosen with care, the quality of the result can still approach the quality level achieved by the best merge algorithm.

Using the preprocessor in combination with Schoenmakers hybrid segmentation method gave poor results for run time as well as the segmentation itself. Using a high minimum polygon size gives an acceptable segmentation result but no run time improvement. A quality assessment using the quality measures is not given because in the hybrid segmentation output, edge addition at the borders of the image is not

cluster threshold	1.0
output format	area marking
cpu time preprocessing	250 sec.
total # of pixels	261117
bmerge thresh.	7
bmerge cpu time	4437 sec.
total cpu time	4687 sec.
segments after ZDM	115743
segments after bmerge	26431
# isol pixels	13289
largest region	89804
# of regions	23744
# of regions > 1 pixel	10358
# of regions with inner pixels	1826

Table 5.10: Best merge results of subjunepart.lan with preprocessing

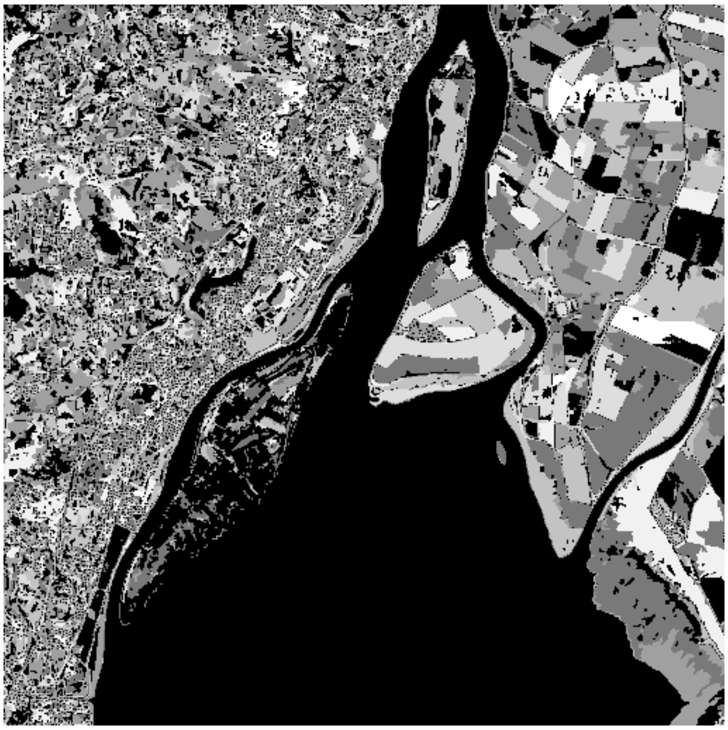


Figure 5.17: Part of the subjune image; clustering with preprocessing followed by best merge

done correctly making the quality values obtained meaningless.

Claims about the quality of the result were based on several sources:

- The quality measures discussed in this chapter.
- Statistical information about the segmentation supplied by the best merge algorithm
- Subjective visual assessment by comparing different segmentation results.

It would be interesting to assess the quality of the segmentation results using other quality measures.

On average 3 segmentations can be made using preprocessing compared to 1 without using preprocessing. Also preprocessing does not cause the optimal best merge threshold to change. The preprocessor can therefore be used when searching for an optimal best merge threshold setting. When an optimal threshold has been found, the segmentation can be done once more using best merge stand alone.

Chapter 6

Conclusions

A preprocessor has been developed for segmentation of satellite images. The preprocessor uses a clustering algorithm to reduce the amount of information present in the satellite image by giving pixels in a cluster the same pixel values. The preprocessing result is made available as an image file derived from the input image file. This output file is then segmented using the best merge algorithm. The preprocessor was developed with the following objective in mind: Speed up the segmentation process without great sacrifice in segementation quality.

6.1 Preprocessor performance

The information reduction achieved by the preprocessor should have a minimal effect on the quality of the segmentation result. This claim was tested in several test cases in chapter 5. In this chapter ways of measuring segmentation quality were discussed. The measurement instruments available were limited to the quality measures described in [TES94] (see section 5.1) together with segmentation statistics given by best merge and visual assessment of the segmentation result.

The test cases showed that if the parameters of the preprocessor are chosen with care, the segmentation result using preprocessing can approach the quality level achieved by the best merge algorithm used stand alone. An example is given in table 5.1. It shows a quality decrease from 0.954 to 0.937, a 2% decrease using preprocessing. There is a trade off between segmentation speed and segmentation quality when using the preprocessor. The use of high clustering thresholds gives good run time performance but causes incorrect merging of segments, shown in section 5.3.3. It should be noted that the results of the best merge algorithm were used as a reference throughout this report. It would be interesting to use another segmentation algorithm and compare the results.

A step in this direction was made in section 5.3.4, where Schoenmakers hybrid segmentation method was used instead of best merge. The results showed that edge detection becomes a problem in a derived image with average values substituted for clustered pixels. Lots of edges are found at the borderline between clustered pixels and non clustered pixels. A segmentation method using edge detection does not combine well with the preprocessor.

The objective of speeding up the segmentation process can be met easily using preprocessing. On average, the test cases showed a 3 fold speed up over using best merge stand alone. This was measured at the segmentation threshold giving the highest segmentation quality. In numbers this means a decrease from 300 to 100 seconds on a 200 x 200 image, see table 5.1 and figure 5.6.

All quality assessments were done using a small set of instruments. Perhaps these are not able to pinpoint the segmentation errors made in the different segmentations well enough to show a clear difference in segmentation quality with and without preprocessing. Different quality measures should be used to further check the quality assessments made in this report.

6.2 Further work

The area finding algorithm described in section 4.5 only scans the image from top to bottom trying to locate the borders of areas. Areas which are concave, "U" shaped or "upside down comb" shaped etc, will be split into several areas. An area linker could be added to find complete areas. This has the following advantages:

- A further reduction in the number of initial segments in the image.
- Speed up of the segmentation.

There are also several disadvantages:

- Using an area linker will produce larger areas in the image with the same average gray levels. This can increase the variance between these areas and neighboring segments or pixels, resulting in more isolated pixels and low merge quality.
- The preprocessing time will increase, making it a less attractive method.

The preprocessor as developed is a stand alone application producing an input image for a segmentation algorithm, best merge in this case. The preprocessor could be integrated with the best merge algorithm using clustering at the segment level. This means that only pixels that are connected can be clustered together. This could produce lower variance between neighboring clustered and non clustered pixels because averages are taken over small areas in the image.

Apart from these enhancements, more experiments could be done using the preprocessor:

- Try other quality measures
- Use an other segmentation algorithm
- Use other images, possibly with ground truth data available.

The results of these experiments can be used for improvement of the preprocessor.

The clustering algorithm used in the preprocessor can be changed to a parallel implementation. Dividing the image data over several machines will give a much faster custering result on large images. Combining parallel clustering with Schoenmakers hybird segmentation method could give a fast parallel segmentation algorithm. This could be done by finding polygons in the original image, mapping them to the preprocessor result.

Appendix A

Images

This appendix contains the satellite images refered to in this document.

Figure A.1: 512 x 512 area from the subjune image shown in figure 4.1. Channel 3 shown

Figure A.2: The subjune image; a 1953 x 1801 Landsat image in 6 spectral bands; from subjune.lan ©Eurimage, band 5 shown

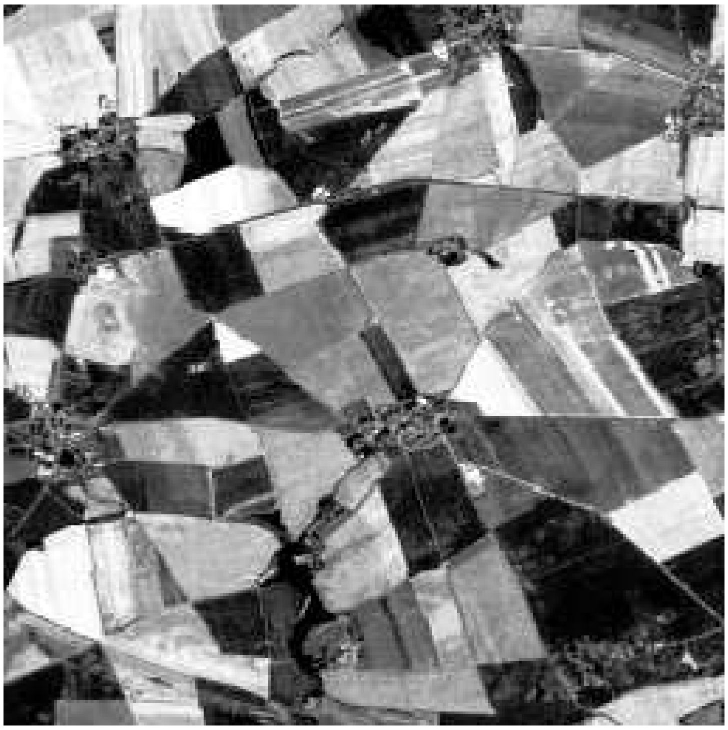


Figure A.3: Channel 3 of the bern image, a 256 x 256 landsat file in 6 bands

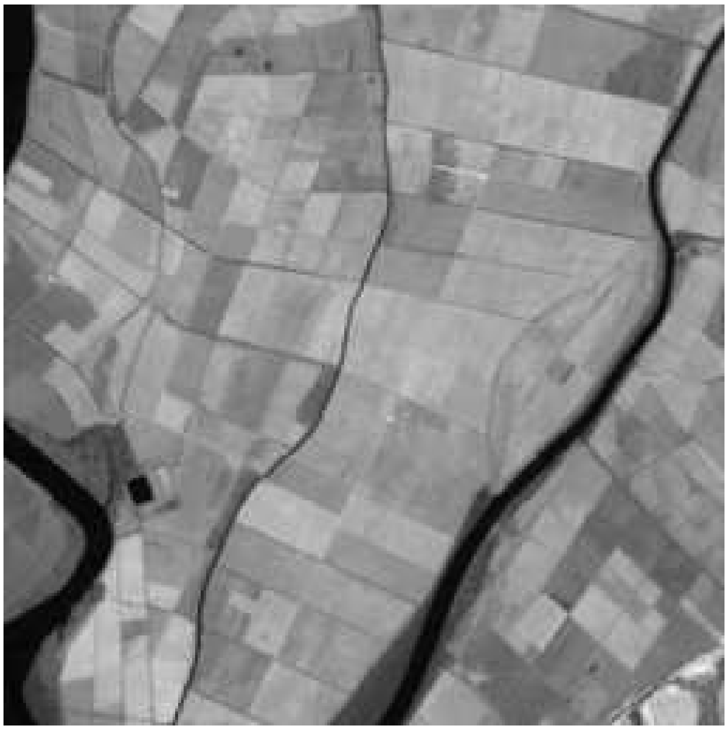


Figure A.4: Suba image, a 256 x 256 area from the subjune image; channel 5 shown.

Bibliography

[BG89] J-M. Beaulieu and M. Goldberg. Hierarchy in picture segmentation: a stepwize optimization approach. In *IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 11 no. 2:150-163*, 1989.

[BL94] Gerardo Beni and Xiaomin Liu. A least biased fuzzy clustering method. In *IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 16 no. 9*, 1994.

[ERD91] ERDAS Inc. Atlanta, GA, USA. *ERDAS Manual, Versions 7.2, 7.4 and 7.5*, 1991.

[For73] M.R. Anderberg Air Force Systems Command United States Air Force. *Cluster Analysis for Applications*. Probability and Mathematical Statistics. Academic Press, New York San Francisco London, 1973.

[Har75] John A. Hartigan. *Clustering Algorithms*. Probability and Mathematical Statistics. Wiley, New York London, 1975.

[HS85] R.M. Haralick and L.G. Shapiro. Survey: Image segmentation techniques. In *Computer, Vision, Graphics, and Image Processing. vol. 29:100-132*, 1985.

[Lac88] Vincaine Lacroix. A three-module strategy for edge detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence. vol.10 no 6.*, 1988.

[MR87] J.M. Meyer-Roux. The ten year research and development plan for the application of remote sensing in agricultural statistics. Technical report, Technical Report SP1.87.39/EN, Commission of European Communities, Joint Research Centre, Institute for Remote Sensing Applications, I-21020, Ispra (Varese), Italy, 1987.

[oaHSM95] Results of a Hybrid Segmentation Method. Quality measures for image segmentation using generated images. In Jacky Desachy, editor, *Image and Signal Processing for Remote Sensing*, Proc. SPIE 2579, 1995.

[Sch83] R.A. Schowengerdt. *Techniques for Image Processing and Classification in Remote Sensing*. Academic Press, 1983.

[Sch95] Ron Schoenmakers. *Integrated Methodology For Segmentation Of Large Optical Satellite Images In Land Applications Of Remote Sensing*. PhD thesis, Computing Science Department, Faculty of Computing Science, Catholic University of Nijmegen, September 1995.

[She95] Ed Sheffner. The landsat program. Technical report, JCWS/Ames Research Center, 1995.

[Smi93] Stephen P. Smith. Threshold validity for mutual neighborhood clustering. In *IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 15 no. 1*, 1993.

[TES94] Maurice S. klein Gebbinck Theo E. Schouten. Quality measures for image segmentation using generated images. In Jacky Desachy, editor, *Image and Signal Processing for Remote Sensing II*, Proc. SPIE 2315, pp 90-101, Rome 1994.

[Til89] J.C. Tilton. Image segmentation by iterative parallel region growing and splitting. In *Proceedings of the International Geoscience and Remote Sensing Symposium (IGARSS89)*, pages 2235–2238, Vancouver, 1989.

[Voo95] Hubert F.J.M. Voogd. Error measurement for segmentation techniques. Master's thesis, Computing Science Department, Faculty of Computing Science, Catholic University of Nijmegen, March 1995.