



RADBOUD UNIVERSITY NIJMEGEN

Color Image Segmentation by Particle Swarm Optimization

Bachelor thesis in Artificial Intelligence

Author:

Agnes VAN BELLE

Supervisors:

Dr. Ida G. SPRINKHUIZEN-KUYPER

Dr. Louis G. VUURPIJL

July 2012

Abstract

This Bachelor thesis will examine the use of the method of Particle Swarm Optimization (PSO) applied to the task of clustering the pixels of an image. In particular, we will examine if the results of the so-called MEPSO algorithm of Das et al. [7] can be replicated for color images. Furthermore, we will look at improvements in both the search space that the swarm moves through and the configuration of the interdependence over time of the particles in the swarm. Our conclusions are that the results of [7] can be replicated for color images, and that certain improvements in both the search space definition and the PSO algorithm itself can be made. Finally, we take a look at how this improved PSO algorithm compares to Soft k-means (SKM) clustering for this task.

Contents

1	Introduction	1
1.1	Swarm Intelligence	1
1.2	Image segmentation	1
1.3	Research question	2
1.4	Outline of this thesis	2
2	Background	3
2.1	Particle Swarm Optimization	3
2.2	PSO and clustering	5
2.2.1	k-means clustering	5
2.2.2	Soft k-means clustering	6
2.2.3	PSO-based clustering	7
2.3	Color image segmentation	9
2.3.1	Color representation	9
2.3.2	Spatial information	14
2.4	MEPSO	15
2.4.1	Spatial window	16
2.4.2	Multi-elitist strategy	16
2.4.3	Clusters with one data point	16
2.4.4	Parameters of MEPSO	17
3	Experimental setup	19
3.1	Choice of color scheme(s)	19
3.1.1	Measuring the color distance	20
3.2	Choice of parameters	21
3.2.1	Acceleration constants	22
3.3	MEPSO implementation choices	23
3.3.1	Spatial window	23
3.3.2	Multi-elitist strategy	23
3.3.3	Clusters with one data point	23
3.4	Choice of input images	24
3.5	Measures used for the results	27

4	Visual results	28
4.1	Comparison with original MEPSO	28
4.1.1	Pepper image	29
4.1.2	Texture image	29
4.2	Color representations	30
4.2.1	Pepper images	30
4.2.2	SimpleBdiff image	31
4.2.3	SimpleSBdiff image	32
4.2.4	Texture image	33
4.2.5	Flower image	34
4.3	Comparison with Soft k-means	35
4.3.1	Pepper images	35
4.3.2	SimpleBdiff image	37
4.3.3	SimpleSBdiff image	37
4.3.4	Texture image	38
4.3.5	Flower image	39
4.4	Parameters	41
4.4.1	Acceleration constants	41
4.4.2	Spatial window	43
5	Data results	45
5.1	Comparison with original MEPSO	45
5.2	Color representations	47
5.3	Comparison to Soft k-means	48
5.4	Acceleration constants	49
6	Discussion	51
6.1	Comparison with original MEPSO	51
6.2	Color representations	51
6.2.1	The D_H distance measure	51
6.2.2	The D_{LSH} distance measure	52
6.3	Comparison to Soft k-means	53
6.3.1	Number of clusters	53
6.3.2	Cluster update rule	53
6.4	Parameters	54
6.4.1	Acceleration constants	54
6.4.2	Spatial window	55
7	Conclusions and future research	56
7.1	Conclusions	56
7.1.1	Summary	56
7.1.2	Comparison with original MEPSO	57
7.1.3	Color representations	57
7.1.4	Parameters	58

7.1.5	Comparison with Soft K-means	58
7.2	Future research	59
7.2.1	Spatial window	59
7.2.2	Run-time complexity	59
7.2.3	Network topology	60
7.2.4	Fitness function	60
7.2.5	Color representation	60
7.3	Concluding remarks	61
A	Code	65
A.1	Conversion between HSL and RGB	65
A.2	HSL distance measures	69

Chapter 1

Introduction

Digital image analysis has many applications such as robotics, security, machine vision and analysis of neural imaging scans. Especially the concept of finding outlines (of objects) in images can be used in these sectors. Examples are helping a robot avoiding objects, or a search engine detecting objects in pictures and finding similar images to the one given. Here we will investigate the application of Swarm Intelligence (SI) to the task of segmenting images.

1.1 Swarm Intelligence

Swarm Intelligence is a discipline which studies decentralized, self-organizing systems, either artificial or biological [1]. An SI system is a multi-agent system in which collective behaviors result from interactions by many homogeneous agents, which themselves can perceive only local information about the environment and are limited to local actions. Biological examples of SI systems are colonies of ants and flocks of birds [21]. Artificial SI systems are inspired by these biological examples. For instance, the fact that ants can leave pheromones on their path, is mimicked in SI approaches for solving combinatorial problems like the travelling salesman problem (this algorithmic technique is called ant colony optimization (ACO), see Dorigo et al. [8]). Artificial SI has also been deployed in data mining algorithms (“ants” picking up and dropping items dependent on the similarity of the item with other items in the ants neighborhood, see Lumer and Faieta [13]). The principles of SI can also be used for solving (continuous or discrete) optimization problems —problems in which the best solution has to be found from all feasible solutions. The subfield of SI used for these kind of problems is called Particle Swarm Optimization (PSO).

1.2 Image segmentation

Partitional cluster analysis is the assignment of elements into subsets such that each subset is as homogeneous as possible, while the subsets between one another, should

be as heterogeneous as possible [11] [9]. The utilization of artificial SI systems for clustering data, in particular the method of PSO, is currently emerging as an alternative to conventional partitional clustering algorithms such as k-means clustering [14] [1]. For the technique of clustering an example can be found in nature too: several ant species cluster corpses (based on their size) to form cemeteries [4] [21].

Image segmentation is the task of identifying homogeneous regions in an image, i.e. clustering the pixels the image is composed of. The task of segmenting images for separating objects, ignoring lighting and texture effects on them, has been extensively investigated since the 1960s [19]. Accomplishments have mainly been made with regard to grayscale images. However, color images convey much more information than grayscale images. When clustering the pixels in a color image, an important aspect of the segmentation method is the color space, from which the color features are inferred. Given a certain color space, there are also various ways to compute the distance between two pixels. An example is the RGB space, with the color distance defined as the Euclidean distance between the R, G and B components.

This thesis examines the application of PSO to the task of image segmentation by examining a proposed PSO-based image segmentation algorithm by Das et al. [7], called MEPSO. As a main advantage over other (image) segmentation or clustering methods is stated that it does not need to (indirectly) assume a priori knowledge of the number of clusters that should be in the resulting partition.

1.3 Research question

Our research question consists of two parts. First and foremost we want to examine if the results of the so-called MEPSO algorithm of Das et al. [7] can be replicated for color images. The second part of our research question is if improvements can be made to that algorithm. For example, improvements tailored to the fact we used color images, or improvements in how the optimization behavior of the swarm is defined.

1.4 Outline of this thesis

In the following Chapter, we will further elaborate on the PSO method and how it can be applied to image segmentation. We will specifically take a look at the PSO-based MEPSO algorithm. In Chapter 3 we will explicate our experimental setup, and describe our proposed improvements for this MEPSO algorithm that we examined. Chapter 4 and Chapter 5 are concerned with the results of our experiments. Finally, Chapter 7 describes our conclusions and some possibilities for future research.

Chapter 2

Background

Before presenting the MEPSO algorithm that we investigated, we will give a brief overview of the techniques and theory it incorporates.

In Section 2.1 we will discuss the general technique of Particle Swarm Optimization (PSO). In Section 2.2 we will look at how PSO can be applied to clustering specifically. Then, in Section 2.3 we will outline some relevant properties of color images to consider when clustering their pixels. Finally, in Section 2.4 we will describe the MEPSO algorithm, which combines the previously discussed techniques by clustering color images using PSO.

2.1 Particle Swarm Optimization

PSO is a computational optimization method that belongs to the field of Swarm Intelligence [1]. PSO is a metaheuristic, as it hardly makes any assumptions about the problem to be optimized. PSO can be applied to any optimization problem. However, it is particularly useful for the optimization of problems for which the goal function is multimodal (having more than one optima) and/or non-linear [18]. Also, PSO does not use the gradient of the problem being optimized, and is therefore suitable for problems whose goal functions are not differentiable, or whose definitions are ill-defined or change over time.

In PSO, the particles that compose the swarm, are each a candidate solution to the problem at hand. Each particle has a fitness value, based solely on its position in the search space [3]. The problem is being optimized by moving the particles through the search space, until a stopping criterion is met. The stopping criterion could be, that all particle's positions don't change more than a certain threshold, or simply that a maximum number of iterations is met. The movement of the particles is based on both their own best position so far and the best position found so far by any particle in their neighborhood. The neighborhood of a particle can consist of the whole swarm, a certain number of particles closest to it in the search space according to some distance measure, or a predefined set of particles (using a connection graph).

Each particle p effectively consists of two vectors: a position vector \vec{Z}_p , and a velocity

vector \vec{V}_p . The position vector is simply the position of the particle in the search space (solution space). The velocity vector denotes the velocity of the particle which defines its movement each iteration. The velocity vector and the position vector are being updated each iteration.

The basic steps of each PSO algorithm are [3]:

1. Evaluate the fitness of each particle;
2. Update the individual and global best fitnesses and positions;
3. Update the velocity and position of each particle.

For a particle p , its velocity \vec{V}_p at time $t + 1$ is updated as follows [12]:

$$\vec{V}_p(t + 1) = \omega \cdot \vec{V}_p(t) + R(0, c_1) \cdot (\vec{P}_{lp}(t) - \vec{Z}_p(t)) + R(0, c_2) \cdot (\vec{P}_{gp} - \vec{Z}_p(t)) \quad (2.1)$$

where:

- ω is the inertia weight ($0 \leq \omega \leq 1$).
- \vec{Z}_p is the vector denoting the position of particle number p .
- \vec{P}_{lp} is the best position particle p has found so far (“local best”).
- \vec{P}_{gp} is the best position found so far by all particles in particle p 's neighborhood (“global best”).
- $R(0, c_i)$ ($i \in \{1, 2\}$) can be written as $c_i \cdot R(0, 1)$, where $R(0, 1)$ is a random diagonal square matrix with each element independently randomly drawn from the uniform distribution in $[0, 1]$.
- c_1 and c_2 are the acceleration coefficients or constants, determining the scale of the movement towards \vec{P}_{lp} and \vec{P}_{gp} .

For a particle p , its position \vec{Z}_p at time $t + 1$ is updated as follows:

$$\vec{Z}_p(t + 1) = \vec{Z}_p(t) + \vec{V}_p(t + 1) \quad (2.2)$$

The updating rule can, alternatively, be written for the m -th dimension (or element) of a particle, as follows:

$$\vec{V}_{pm}(t+1) = \omega \cdot \vec{V}_{pm}(t) + \text{RAND}(0, c_1) \cdot (\vec{P}_{lpm} - \vec{Z}_{pm}(t)) + \text{RAND}(0, c_2) \cdot (\vec{P}_{gpm} - \vec{Z}_{pm}(t)) \quad (2.3)$$

where $\text{RAND}(0, c_i)$ is a random number in $[0, c_i]$.

The parameter ω can be seen as the amount of resistance to a change in movement of the particles. The vector $\vec{V}_p(t)$ can be seen as the momentum of the particle, a force that maintains that each particle can only depart with regard to its current direction. The vector $\vec{P}_{lp}(t)$ can be seen as a cognitive or memory component, the tendency for a particle to take into account the value of its own best position found so far. Finally, the vector \vec{P}_{gp} can be seen as the social component —the tendency for a particle to take into account the best solution found so far by the other particles in its neighborhood.

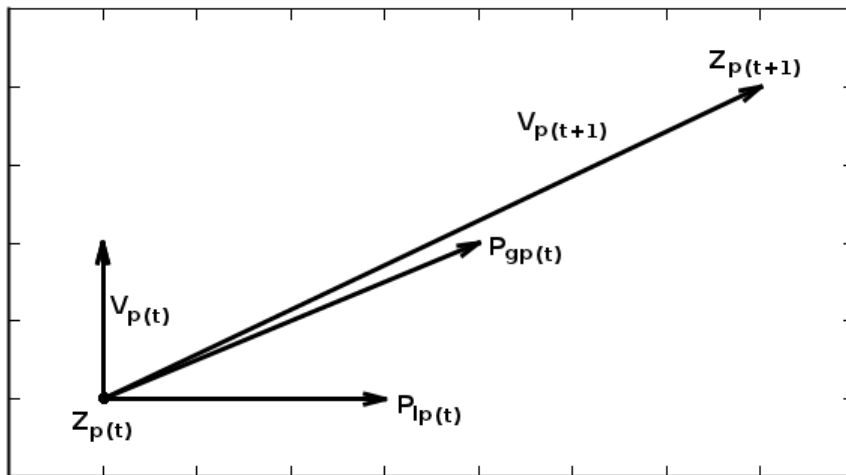


Figure 2.1: The movement process illustrated for a two-dimensional search space, $\omega = 1$ and $R(0, c_i) = 1$.

The values in \vec{V}_p (for particle p) are usually kept within a certain range, to avoid exploding velocities. However, this does not necessarily prevent the particles from moving beyond the search space boundaries. To rule this possibility out completely, one can also clamp the values of the position vectors, \vec{Z}_p (for particle number p).

2.2 PSO and clustering

Cluster analysis is concerned with the division of objects (data vectors) into subsets, such that each subset is as homogeneous as possible, while each subset should differ as much as possible from other subsets. First we will take a look at two very common non-PSO clustering algorithms, k-means clustering and fuzzy k-means clustering. Then we will look at how PSO can be useful for clustering.

2.2.1 k-means clustering

A popular partitional clustering algorithm is the k-means algorithm [11]. In this procedure, there are k clusters, each represented by a cluster centroid \vec{C}_j , $j \in \{1, 2, \dots, k\}$. These cluster centroids have to be initialized at start. Every iteration, each data point

is assigned to the centroid nearest to it. Cluster centroids then are updated as to become the mean of the data points assigned to them. This process is repeated, and the algorithm terminates when the centroids no longer move.

The goal of the algorithm is to minimize a squared error function, the objective function J :

$$J = \sum_{c=1}^k \sum_{i=1}^{n^{(c)}} \|\vec{X}_i^{(c)} - \vec{C}_c\|^2 \quad (2.4)$$

where $\vec{X}_i^{(c)}$ is the i -th data vector assigned to cluster number c , \vec{C}_c is the cluster centroid of cluster c , and $\|*\|$ is any norm expressing the similarity between the data vector and cluster centroid vector.

The most obvious drawbacks of k-means are that the number of clusters needs to be specified beforehand, and that the result is dependent on the initialization of the centroids (for a short overview and one example of initialization methods, see [17]). Also, there is no guarantee that it will converge to the global optimum. Often, the algorithm is run multiple times with different initial conditions.

2.2.2 Soft k-means clustering

The soft k-means clustering algorithm (SKM) is an extension of the k-means algorithm. In essence, it is a fuzzification of the k-means algorithm [10]. It is also known under the more common name of fuzzy c -means (FCM), and is the most popular fuzzy clustering algorithm [11]. As with the k-means algorithm, in the SKM algorithm there are k clusters, each represented by a cluster centroid \vec{C}_c , $c \in \{1, 2, \dots, k\}$, and these cluster centroids have to be randomly initialized at start.

Every iteration, each data point is assigned with a certain degree to each centroid. That is, the assignment is not crisp —each data point \vec{X}_i has a membership degree u_{ci} for every cluster \vec{C}_c . This leaves us with an $n \cdot k$ matrix \mathbf{U} , at each iteration, where the number of rows n is the number of data points and the number of columns k is the number of cluster centroids considered.

Cluster centroids then are updated to become the weighted mean of the data points assigned to them, where the weight for each data point consists of its membership grade to that cluster. That is, the mean of all data points, weighted by their degree of belonging to the cluster (see Equation 2.5).

The level of fuzziness of the clustering algorithm is influenced by a variable m ($1 \leq m \leq \infty$), the “fuzzifier”. This is the power to which the membership degree u_{ci} is raised when minimizing the objective function (see Equation 2.5). The larger m is, the smaller the membership degrees, hence the smaller the differences between the levels of membership [10] [6]. Approaching m ’s lower limit of $m = 1$, the algorithm approaches a crisp clustering algorithm, with the membership degrees u_{ci} converging to 1 or 0 —that is, it converges to the k-means algorithm. Approaching m ’s upper limit of $m = \infty$, the membership degrees for all data points, with regard to all k clusters, converges to $\frac{1}{k}$. In absence of knowledge about the domain, m is usually set to 2.

The objective function to be minimized is:

$$J_m = \sum_{c=1}^k \sum_{i=1}^n u_{ci}^m \cdot \|\vec{X}_i - \vec{C}_c\|^2 \quad (2.5)$$

Cluster centroids can be updated as follows:

$$\vec{C}_c \leftarrow \frac{\sum_{i=1}^n u_{ci}^m \cdot \vec{X}_i}{\sum_{i=1}^n u_{ci}^m} \quad (2.6)$$

The degree of membership of the i -th data point to the c -th cluster, u_{ci} , is defined as follows [6]:

$$u_{ci} = \frac{1}{\sum_{l=1}^k \left(\frac{\|\vec{X}_i - \vec{C}_c\|}{\|\vec{X}_i - \vec{C}_l\|} \right)^{\frac{2}{m-1}}} \quad (2.7)$$

Equation 2.7 follows if one solves

$$\forall i \in [1, n], \forall c \in [1, k], \frac{\partial J_m}{\partial u_{ci}} = 0$$

and

$$\forall c \in [1, k], \frac{\partial J_m}{\partial \vec{C}_c} = 0$$

(see [1] and [15]).

Again, drawbacks are that the number of clusters needs to be specified beforehand, and that the result is dependent on the initial selection of the centroids. There is again no guarantee that it will converge to the global optimum.

2.2.3 PSO-based clustering

The utilization of PSO for clustering data has not been researched much. Any PSO-based algorithm is absent in the surveys of Jain et al. [11] (1999) and Elavarasi et al. [9] (2011). Applying PSO to fuzzy clustering for clustering image pixels is proposed by Das et al. [7] as an alternative to conventional partitional clustering algorithms (for example, based on evolutionary algorithms, or non-PSO adaptations of SKM-clustering). According to this paper, applying PSO to fuzzy clustering has not yet been conducted.

As said, in PSO, each particle must represent a possible solution to the problem. PSO can be used as a method for clustering by letting each particle represent k cluster centroids:

$$\vec{Z}_p = \boxed{\vec{C}_{p,1} \quad \vec{C}_{p,2} \quad \dots \quad \vec{C}_{p,k}} \quad (2.8)$$

That is, the position vector \vec{Z}_p of each particle, consists of k vectors, called $C_{p,k}$, which each represent a cluster centroid.

An important advantage of using PSO over traditional cluster algorithms, is that it can maintain, recombine and compare several candidate solutions simultaneously. Therefore PSO is good at coping with local optima. In contrast, k-means clustering always will converge to the local optimum that is the closest to the starting point of the search.

A dynamic amount of clusters

Another important advantage is that when using PSO one does not have to assume a priori knowledge of the amount of clusters. This can be accomplished by first determining a maximum number of possibly active clusters, c_{max} . Then one includes, in the position vector \vec{Z}_i , activation values for each cluster centroid:

$$\vec{Z}_p = \boxed{a_{p,1} \quad a_{p,2} \quad \dots \quad a_{p,c_{max}} \quad | \quad \vec{C}_{p,1} \quad \vec{C}_{p,2} \quad \dots \quad \vec{C}_{p,c_{max}}} \quad (2.9)$$

When an activation value $a_{p,c}$ ($c \in \{0, 1, \dots, c_{max}\}$) is below a certain activation threshold T , its corresponding cluster centroid $C_{p,c}$ is disregarded from all clustering computations—it can not have any members and does not play a role in any objective function or fitness function applied to the particle. We assume that the activation values are initialized randomly, and we will assume that the activation values are in $[0, 1]$, and that the activation threshold, T , is 0.5.

Incorporating this approach results in an extension of the dimensionality of the search space the particles move through. Now, the position vector \vec{Z}_p as well as the velocity vector \vec{V}_p are of length $c_{max} + c_{max} \cdot d$ (where d is the number of dimensions of the search space or feature space). Incorporating this extension in the velocity updating process requires no extension of the updating rule (Equation 2.1):

$$\vec{V}_p(t+1) = \omega \cdot \vec{V}_p(t) + R(0, c_1) \cdot (\vec{P}_{lp}(t) - \vec{Z}_p(t)) + R(0, c_2) \cdot (\vec{P}_{gp} - \vec{Z}_p(t))$$

However, recall from Section 2.2 that the values in \vec{V}_p are usually kept within a certain range. These upper and lower bounds will be different for the activation thresholds. For example, the velocity values corresponding to the activation thresholds could be kept within $[-1, 1]$. We will use a different notation to distinguish between the velocity bounds corresponding to the cluster centroids values in \vec{V}_p , and the velocity bounds corresponding to the activation values in \vec{V}_p (see Table 3.1 on page 21).

The fitness function

A PSO algorithm needs a fitness function. This fitness function is needed by each particle to determine its own best position so far, $\vec{P}_{lp}(t)$, and the best position found so far by any particle in its neighborhood, $\vec{P}_{gp}(t)$. A fitness function for a PSO algorithm

concerned with clustering has to be based on an appropriate cluster validity index. The fitness function Das et al. [7] use is based on the Xie-Beni index [22]. The smaller this index, the better the solution. This fitness function is appropriate for any fuzzy clustering algorithm.

The Xie-Beni index is defined as follows. First we define the deviation of a data point X_i from cluster c , as $d_{ci} = u_{ci} \cdot \|\vec{X}_i - \vec{C}_c\|$. The variation of a cluster c is defined as $\sigma_c = \sum_{i=1}^n d_{ci}^2$. The summation of the variations of all clusters, is the total variation of the data set with respect to the clustering partition, and denoted by $\sigma = \sum_{c=1}^k \sigma_c$. This total variation should, of course, be as low as possible. However, for the total variation to have a valid meaning for each data set, the cluster validity index should also take into account the number of data points clustered. Therefore we divide σ by the number of data points: $\frac{\sigma}{n}$. Finally, the validity index should also be proportional to the distance between cluster centroids. The between-cluster distance should be as maximal as possible in any clustering solution. Therefore $\frac{\sigma}{n}$ is divided again by the minimum between-cluster distance in the partition, $D_{\min} = \min_{c \neq d} \|\vec{C}_c - \vec{C}_d\|^2$.

The index has now become $\frac{\sigma}{(n \cdot D_{\min})}$. This can be written as

$$XB = \frac{\sum_{c=1}^k \sum_{i=1}^n u_{ci}^2 \cdot \|\vec{X}_i - \vec{C}_c\|^2}{n \cdot \min_{c \neq d} \|\vec{C}_c - \vec{C}_d\|^2} \quad (2.10)$$

where \vec{C}_c is the c -th cluster centroid, and \vec{X}_i is the i -th data point.

Note that the numerator in this fraction (σ) is equal to the objective function to be minimized used in soft k-means clustering, when $m = 2$ (see Equation 2.5).

The fitness function that has to be maximized for each particle p , is, accordingly:

$$f_p = \frac{1}{XB_p + \epsilon} \quad (2.11)$$

where ϵ is a very small constant (e.g. 0.0001) to avoid that the denominator will become zero.

2.3 Color image segmentation

In this Section we will discuss two important aspects pertaining to the segmentation of color images: the color representation used, and the incorporation of spatial information.

2.3.1 Color representation

In a grayscale image, the difference between two pixels can simply be measured as the difference in brightness between these two pixels, because that is the definition of a difference between two shades of gray. For color images, only considering the brightness

is not enough, since two distinct colors can be of the same brightness. There are different color representations, or spaces, with which one can represent an image. Below we describe RGB, HSI, HSL, and HSV.

RGB

RGB stands for Red, Green and Blue. These three colors are combined to form all colors. The RGB color space can be represented by a cube:

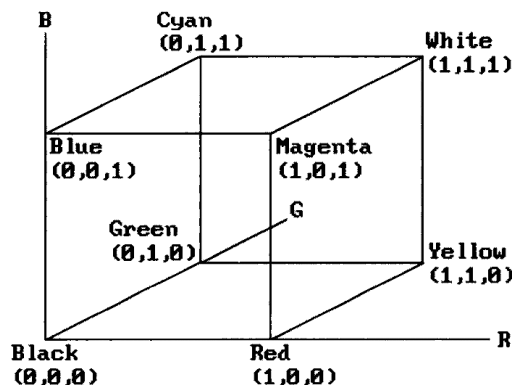


Figure 2.2: The RGB color space (taken from [2]).

In Figure 2.2, the value range for each component is denoted as being $[0, 1]$. In reality, the range is often specified as the number of values possible for each component, which depends on the number of bits each component has been assigned to use. For example, when using a monitor which uses the Truecolor (24-bits per color) color depth, each component has 8 bits available to it, and as such, the range for each component is usually denoted as being $[0, 255]$. The distance between two RGB values is usually defined as the Euclidian distance of two points in the RGB cube.

RGB is widely used for color display, for example in LCD and plasma TV screens, and for all computer and mobile phone displays. RGB is not very suitable for color analysis and segmentation. If, for example, the “brightness” of a color changes, all three R, G and B values will change accordingly. That is, there is a high correlation between the three components.

HSI

HSI stands for Hue, Saturation and Intensity. It is not a standardized definition. It can be thought of as comprising both the Hue, Saturation, Lightness and the Hue, Saturation, Value color model, which are more strictly defined, and which we will address below.

Similar to the RGB model, in the HSI model a color is also defined by three components. The HSI coordinates (i.e. the HSL or HSV coordinates) can be derived for any color in the RGB space, and vice versa. However, the three components themselves

are very different in the HSI model. Instead of a red (R), a green (G) and a blue (B) component, one discerns the “pure color” (H, the hue); the degree of colorfulness (S, the saturation), and the brightness (I, the intensity).

Hue is determined by the dominant wavelength in the optical spectrum. It solely comprises the colors one can break light into (for example, with a prism). The hue value is in an angular range, from 0° to 360° . For example, pure yellow is at 60° , pure blue is at 240° . Pure red is at both 0° and 360° . The saturation and intensity values are not represented as angular, but as scalars. The intensity component can be thought of as the brightness. Set to zero intensity a picture will be fully black, set to full intensity a picture will be fully white. The saturation component refers to the purity of the hue, or the “vividness” of the hue. It signifies the amount of the intensity, mixed with the hue, where a lower saturation means more of the intensity is mixed with the hue.

This effectively means that, when saturation is set to zero, the picture will appear in all gray tones. (Note that this leaves the hue component undefined.) The tone or “darkness” or “lightness” of the gray tones, however, will depend on the intensity component. The HSI color model can be represented by a cylinder:

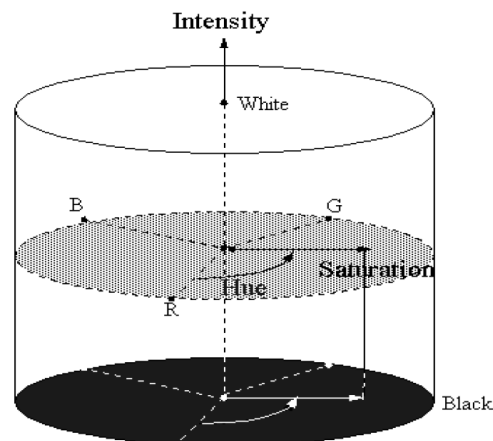


Figure 2.3: The HSI color space (taken from [5]).

Separate representations for color (hue) on one hand, and intensity and saturation on the other hand, can be very useful for separating the objects in a picture, regardless of highlights, shadows, shadings or textures.

The drawback with this model is that when saturation diminishes, the radius around which hue revolves diminishes too. Therefore, hue becomes more unstable. That is, the small changes in perceivable color that one can get when having a low saturation, will be represented by the same amount of hue difference in degrees, as large changes in perceivable color, when having an average saturation. To deal with this, many segmentation algorithms leave pixels with low saturation unassigned to any cluster [5].

HSV

HSV stands for Hue, Saturation and Value, and is a variant of HSI. Instead of the intensity component, I, a value component, V, is used.

It should be noted that HSV is also known as HSB: Hue, Saturation, Brightness. Both represent the same color space. The term Value is a transformation of the term Intensity such, that the lower the value, the less difference in saturation is possible. That is, when the value is minimal, the saturation is not defined and therefore also the hue is not. However, when the value is maximal, the saturation still ranges from pure white to the pure hue.

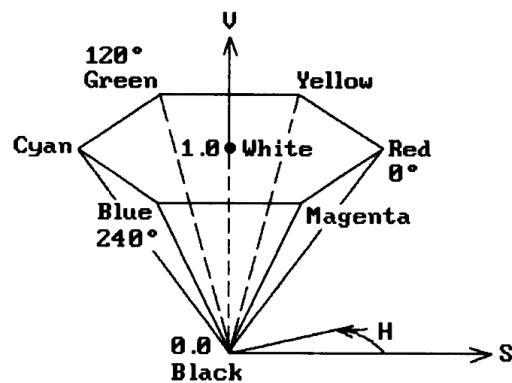


Figure 2.4: The HSV color space (taken from [2]).

HSV can be thought of as the color space of a painter's palette. If you start with red paint, you can easily get a pure black color by mixing it with black paint (i.e. decreasing the value component). The hue range will decrease with every drop of black paint. However, you can never get a pure white color by mixing the red paint with white paint (i.e. increasing the value component), no matter what copious amount of white paint. For getting a pure white color, you should not have started with any red paint at all (i.e. left the saturation at zero).

HSL

HSL stand for Hue, Saturation and Lightness, and is also a variant of HSI. Instead of the intensity component, I, a lightness component, L, is used.

The term Lightness is a transformation of the term Intensity, such, that the more the lightness approaches its minimum or maximum, the less difference in saturation is possible. So, as an additional restriction compared to the HSV model, when the value is maximal, the saturation is also undefined. This means that when the intensity of the color is minimal or maximal, both hue and saturation play no role in distinguishing colors.

HSL best represents how humans perceive color gradations found in nature. Contrary to a painter's palette, in nature a very high lightness (or "brightness") can render the

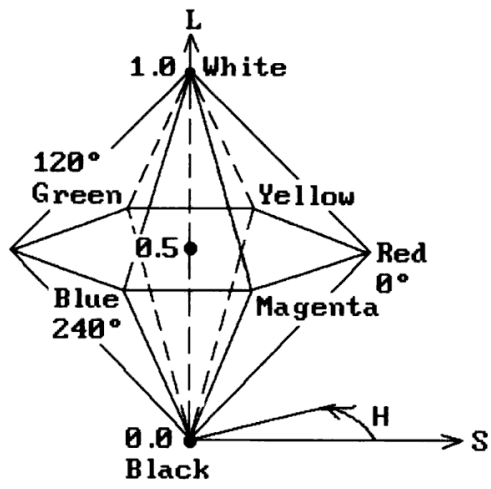


Figure 2.5: The HSL color space (taken from [2]).

most vivid, colorful object as pure white.

2.3.2 Spatial information

The goal of image segmentation is dividing an image in homogeneous regions. Regarding this as a clustering problem, each image region, where the smallest possible region is one pixel, will be assigned to a certain cluster. When clustering the data of an image, there are already spatial hints for deciding to which cluster a pixel belongs, contrary to when clustering non-image data. Naturally, it will often be the case that pixels next to each other will belong to the same cluster. This gives us an additional clue to which cluster a pixel should be assigned.

To exploit this spatial information, Das et al. [7] introduce a method that can be thought of as centering a square window on a pixel i , and taking into account the membership degree to cluster c of each of those pixels in the square window, when determining the membership grade of pixel \vec{X}_i to cluster \vec{C}_c .

The function that takes into account the membership degree to cluster \vec{C}_c of each of those pixels in the square window on pixel \vec{X}_i , is called h_{ci} . The value of h_{ci} is simply the sum of the membership degrees to cluster \vec{C}_c of all other pixels in the window surrounding pixel \vec{X}_i . It represents a new membership degree of pixel \vec{X}_i to cluster \vec{C}_c , solely based on the spatial location of pixel \vec{X}_i .

This spatial function h_{ci} is defined as

$$h_{ci} = \sum_{q \in \delta(\vec{X}_i)} u_{cq} \quad (2.12)$$

where $\delta(\vec{X}_i)$ represents the square window centered on pixel \vec{X}_i , and u_{cq} is the degree of membership of pixel \vec{X}_q to cluster \vec{C}_c .

Now the general fuzzy clustering formula for determining the degree of membership of a pixel to a cluster must incorporate this new membership function. Das et al. [7] propose the new formula as

$$u'_{ci} = \frac{u_{ci}^r \cdot h_{ci}^t}{\sum_{d=1}^k u_{di}^r h_{di}^t} \quad (2.13)$$

The values for r and t in Equation 2.13 will determine the relative importance of the spatial information. In this paper r and t will both be set to 1, just as in the original MEPSO paper of Das et al. [7]. So the formula simply becomes

$$u'_{ci} = \frac{u_{ci} h_{ci}}{\sum_{d=1}^k u_{di} h_{di}} \quad (2.14)$$

To acquire an intuitive understanding of Equation 2.14, recall from Section 2.2.2 (page 6) that u_{ci} represents the probability that pixel number i belongs to cluster number c , based on the distance between pixel \vec{X}_i and cluster centroid \vec{C}_c . In the numerator of the fraction of Equation 2.14, h_{ci} is multiplied by u_{ci} , and as such a new combined

membership degree or probability is computed. We can call this combined probability $m_{ci} = u_{ci} \cdot h_{ci}$. This number m_{ci} is then divided by the sum of the m_{di} 's for all clusters \vec{C}_d ($d \in \{0, 1, \dots, k\}$), $\sum_{d=1}^k m_{di}$, to distribute the new membership degree u'_{ij} evenly per cluster, that is, to make $\sum_{i=1}^k u'_{ij} = 1$. Using this formulation, Equation 2.14 can be written as

$$u'_{ci} = \frac{m_{ci}}{\sum_{d=1}^k m_{di}} \quad (2.15)$$

The square window can of course be of any reasonable size. Das et al. [7] use a window of 5×5 pixels.

An example window is shown in Figure 2.6. Here, the selected pixel \vec{X}_i is on the corner of a square, and its color is the same as the color of the square, that is, the lightest color in this image. It is partly surrounded by pixels of the lighter square, and partly by pixels of the darker square. This pixel \vec{X}_i should originally, before the spatial function is applied, have the same membership degrees u_{ci} ($c \in \{0, 1, \dots, k\}$) for all clusters as the rest of the pixels of the lighter square. However, after the spatial function is applied, it will have a lower membership degree for the cluster that the pixels of the lighter square have the highest probability to belong to, and a higher membership degree for the cluster that the pixels of the darker square have the highest probability to belong to, because 16 pixels in its window belong to the darker square, and only 8 to the lighter square. Thus, the corner of the lightest square might become rounded.

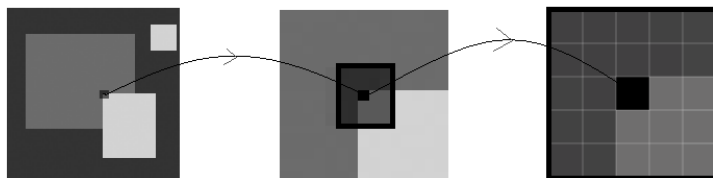


Figure 2.6: A 5×5 pixels window (surrounded by a black border) placed on a pixel (colored black for this purpose) shown in the same image three times. In the first image the total image is shown; in the second image we zoom in on the window; in the third image we zoom in further and acquire a complete view of the window at full pixel resolution.

2.4 MEPSO

MEPSO stands for Multi-Elitist Particle Swarm Optimization and is adaptation of the general PSO method, proposed by Das et al. [7].

An important setting is that they let the social neighborhood of a particle consist of all other particles. That means that the global best position is the same for each particle

in an iteration.

The creators of MEPSO also introduce a few new things to the general PSO method: a spatial window, a multi-elitist strategy, and a way to deal with clusters with less than two members.

2.4.1 Spatial window

The concept of a spatial window to exploit the fact that neighboring pixels will often show a high degree of correlation has been already discussed in Section 2.3.2. As pointed out there, the values for r and t in the formula for u'_{ij} , Equation 2.13, will determine the relative importance of the spatial information. In the original experiments with the MEPSO algorithm, r and t were assumed to be 1, as in Equation 2.14.

2.4.2 Multi-elitist strategy

MEPSO is called multi-elitist because when determining the new global best at each iteration, it does not simply make the position with the highest fitness value the new global best. Instead, it first selects all particles with a position better than the global best so far, puts them in a “candidate area”, and then makes the position of the particle in the “candidate area” with the highest growth rate the new global best. The growth rate of a particle starts at 0, and is increased with 1, for every iteration its fitness value is higher than the fitness value it had on its previous iteration. That is, this multi-elitist strategy prefers the solutions of steadily “hill-climbing”-particles above those of particles that strongly fluctuate.

The pseudo-code for the selection of the global best, incorporating the multi-elitist strategy, is given in Algorithm 1.

2.4.3 Clusters with one data point

As defined in Equation 2.7 (page 7), the membership degree of a data point/particle is defined as follows:

$$u_{ci} = \frac{1}{\sum_{l=1}^k \left(\frac{\|\vec{X}_i - \vec{C}_l\|}{\|\vec{X}_i - \vec{C}_c\|} \right)^{\frac{2}{m-1}}}$$

It is possible that an active cluster \vec{C}_k has exactly only one data point in it. If this data point \vec{X}_i would have a membership degree of 1 to that cluster centroid, \vec{X}_i will be equal to \vec{C}_k . This means a division by zero will occur in the above formula when calculating the membership degrees u_{ci} for data point number i .

In the MEPSO algorithm this is resolved by checking each iteration if any active cluster centroid has only one data point in it. If so, the cluster centroid is re-initialized. It is, however, unclear how this re-initialization exactly happens. We quote: “*the cluster center positions of this special chromosome are re-initialized by an average computation. We put $\lfloor n/k \rfloor$ data points for every individual cluster center, such that a data point goes*

Algorithm 1 The multi-elistist strategy pseudocode

```
for  $t = 1 \rightarrow t_{max}$  do
  if  $t < t_{max}$  then
    for  $j = 1 \rightarrow popSize$  do
      if  $fitness(particle_p) > fitness(particle_p \text{ in } (t - 1)_{th} \text{ time step})$  then
         $beta_j = beta_j + 1$ 
      end if
      update localBest $j$ 
      if  $fitness(localBest_p) > globalBest$  then
        put localBest $p$  in candidateArea
      end if
    end for
    candidateHighestBeta  $\leftarrow$  particle in candidateArea with highest beta
    newGlobalBest  $\leftarrow$  candidateHighestBeta
    globalBest  $\leftarrow$  newGlobalBest
  end if
end for
```

with a center that is nearest to it” [7]. This seems to presume a priori knowledge, that is, knowledge of which cluster centroid configuration is needed given the data. Also, the authors seem to suggest that every cluster centroid of the particle is re-initialized, while only one cluster centroid is erroneous.

Our method to resolve this issue is different and will be outlined in Section 3.3.3.

2.4.4 Parameters of MEPSO

There are various parameters to consider when implementing a MEPSO-based PSO algorithm, outlined in Table 2.1.

When $c_1 > 0$ and $c_2 = 0$, the particles do not take into account any fitness information by other particles and are independent hill-climbers. When $c_1 = 0$ and $c_2 > 0$, a particle does not take into account its own fitness history and the swarm is one stochastic hill-climber. For multimodal problems, i.e. when multiple solutions need to be found, it is best to set $c_1 > c_2$ [16]. Also, intuitively, low values for c_1 and c_2 result in smooth particle movements, while high values for c_1 and c_2 result in more abrupt movements and a higher chance for particles to leave the search space (the value limits on the velocity and cluster components, like \vec{V}_{max} , \vec{Z}_{max} etc., can be used avoid this). The acceleration constants could also be increased or decreased over time.

Because clusters (cluster centroids) can be set to active or inactive by the activation thresholds, a minimum number of active clusters should be specified. The maximum number of active clusters is equal to the total possible number of cluster centroids.

The maximum number of iterations (*itCount*), is only relevant if the iteration count is the stopping criterion. An alternative is to define a certain threshold below which all particles’ change in position must be for the algorithm to terminate.

Description	Name	Domain
Inertia weight	ω	$[0, 1]$
Acceleration constants	c_1, c_2	$[0, \infty]$
Minimal and maximal number of clusters	$cmin, cmax$	$[2, \infty]$
Population size	$popSize$	$[1, \infty]$
Number of iterations	$itCount$	$[1, \infty]$
Maximum velocity value for positions	\vec{V}_{max}	Depends on data
Minimum velocity value for positions	\vec{V}_{min}	$-\vec{V}_{max}$
Maximum velocity value for activation values	V_{max_a}	Free choice
Minimum velocity value for activation values	V_{min_a}	Free choice
Activation threshold for activation values	T	Free choice
Maximum positional coordinates	\vec{Z}_{max}	Depends on data
Minimum positional coordinates	\vec{Z}_{min}	Depends on data
Maximum coordinate value for activation values	Z_{max_a}	Free choice
Minimum coordinate value for activation values	Z_{min_a}	Free choice
Size of the spatial window	$windowSize$	$[1, x]$ where $x = \text{odd}$
Neighborhood topology and size of the swarm	/	global, $[1, popSize - 1]$

Table 2.1: MEPSO-based PSO parameter overview

Chapter 3

Experimental setup

The main goal of this research is to examine if the results of Das et al. [7] obtained with their MEPSO algorithm can be replicated, after which we will elaborate on possible improvements.

3.1 Choice of color scheme(s)

All pictures used for examination in this thesis will first be processed as represented by the RGB color scheme, with each of the three components represented by 8 bits. That is, the image will have a 24-bit color depth (Truecolor, $256^3 = 16777216$ possible colors).

The original MEPSO algorithm is tested with such an RGB representation of the image data. In this thesis our own MEPSO implementation will also be examined with the data of the image represented by the HSL color model.

HSL is chosen above HSV, because in the HSL model, saturation is undefined when the lightness is both maximal (white) and minimal (black), while in the HSV model, the saturation is still defined when the lightness (value) is maximal. The first property resembles human color vision best. See also Section 2.4.1.

The prediction is that by using the HSL color representation with a tailored similarity measure we can improve the image segmentation process. For example, the distance or difference between two data points (pixels) could e.g. be calculated as being more dependent on the hue (i.e. the color itself) than on the saturation and lightness (i.e. highlight, shadings and shadows).

To convert the input images from RGB to HSL and vice versa (to obtain the resulting picture), conversion algorithms from Agoston [2] are used (see Appendix A.1). An important property of these algorithms is how to deal with pixels with an undefined (resulting) hue value, which occurs if the pixel representation is pure grayscale, i.e. its saturation component is zero. When converting from RGB to HSL, an undefined hue component is being represented in the HSL representation as the value Not a Number (*NaN*). When converting from HSL to RGB, if a pixel has an undefined hue value, only its lightness component will be used in the rest of the conversion algorithm.

3.1.1 Measuring the color distance

To measure the distance between a data point, i.e. a pixel, and a cluster centroid, we need to have an appropriate measure for each color representation used.

When using the RGB color scheme, the distance can simply be calculated as the Euclidean distance, since the RGB color scheme can be represented by a cube (see Section 2.3.1).

Using the HSL color scheme, the choice of a distance measure is less straightforward. First, one could weigh each color component (H, S and L) equally, or place more (or all) emphasis on the hue component, to avoid segmenting the more shadowed and more illuminated parts of an object separately. Second, one has the problem of the hue component being undefined when the saturation component is zero. As pointed out in Section 2.3.1, many segmentation algorithms simply ignore these pixels in the clustering process.

Here, we tested the algorithm with two separate distance measures for calculating the color difference under the HSL representation. One measure utilizes just the hue component. We will call this measure D_H (see Equation 3.2). When one of the hue components is undefined, the distance between the two data points is simply deemed to be zero.

The second measure, D_{LSH} (see Equation 3.4), is based on the work of Patrascu [15], which proposed a HSL similarity measure that takes into account all three HSL color components. Here, the emphasis that is put on both the hue and lightness component, is dependent on the value of the saturation component. The higher the saturation value, the more emphasis is put on the hue component and the less emphasis is put on the lightness component and vice versa.

To define the equations, first we need separate notations for the hue, saturation and lightness components of a pixel in the HSL color representation. Let $h_i \in [0, 360]$ denote the hue component of pixel \vec{X}_i . Similarly, $s_i \in [0, 1]$ denotes the saturation component of pixel \vec{X}_i , and $l_i \in [0, 1]$ denotes the lightness component of pixel \vec{X}_i .

Thus, we define the HSL component distances between the pixels \vec{X}_i and \vec{X}_j as:

$$\begin{cases} d_h(i, j) = \min(|h_i - h_j|, 360 - |h_i - h_j|) / 180 \\ d_s(i, j) = |s_i - s_j| \\ d_l(i, j) = |l_i - l_j| \end{cases} \quad (3.1)$$

Given these definitions, we define the D_H distance measure as:

$$D_H(\vec{X}_i, \vec{X}_j) = \begin{cases} 0 & \text{if } h_i \text{ or } h_j \text{ is UNDEFINED} \\ d_h(i, j) & \text{otherwise} \end{cases} \quad (3.2)$$

For the D_{LSH} distance function, we first define the chromaticity r_i and achromaticity a_i for a pixel \vec{X}_i :

$$\begin{cases} r_i = \sin(s_i \cdot (\pi/2)) \\ a_i = \cos(s_i \cdot (\pi/2)) \end{cases} \quad (3.3)$$

We now can define the D_{LSH} distance function as:

$$D_{\text{LSH}}(\vec{X}_i, \vec{X}_j) = \begin{cases} 0 & \text{if } h_i \text{ or } h_j \text{ is UNDEFINED} \\ a_i a_j \cdot d_l(i, j) + r_i r_j \cdot d_h(i, j) & \text{otherwise} \end{cases} \quad (3.4)$$

One can consult Appendix A.2 for the exact Matlab code.

3.2 Choice of parameters

A parameter overview of the original MEPSO experiments is outlined in Table 3.1.

Description	Name	Value
Inertia weight	ω	0.794
Acceleration constants	c_1	0.35 \rightarrow 2.4
	c_2	2.4 \rightarrow 0.35
Minimal and maximal number of clusters	$cmin$	2
	$cmax$	10
Population size	$popSize$	40
Number of iterations	$itCount$	unclear
Maximum velocity value for positions	V_{max}	255
Minimum velocity value for positions	V_{min}	-255
Maximum velocity value for activation values	V_{max_a}	1
Minimum velocity value for activation values	V_{min_a}	-1
Activation threshold for activation values	T	0.5
Maximum positional coordinates	\vec{Z}_{max}	Unclear if used
Minimum positional coordinates	\vec{Z}_{min}	Unclear if used
Maximum coordinate value for activation values	Z_{max_a}	Unclear if used
Minimum coordinate value for activation values	Z_{min_a}	Unclear if used
Size of the spatial window	$windowSize$	5
Neighborhood topology and size of the swarm	/	global, $[1, popSize - 1]$

Table 3.1: Original MEPSO parameter overview [7]

Note that from the original MEPSO paper it can not be determined exactly how the acceleration constants are being gradually increased and decreased, partly because their exact number of iterations for each run is also unclear.

As can be seen in Table 3.2, most parameters were kept the same in our implementation, except the population size and the acceleration constants.

Description	Name	Value
Inertia weight	ω	0.794
Acceleration constants	c_1	1
		2 \rightarrow 0.1
		0.1 \rightarrow 2
	c_2	1
		0.1 \rightarrow 2
		2 \rightarrow 0.1
Minimal and maximal number of clusters	$cmin$	2
	$cmax$	10
Population size	$popSize$	10
Number of iterations	$itCount$	20
Maximum velocity value for positions	V_{max}	255 (RGB)
		1 (HSL)
Minimum velocity value for positions	V_{min}	-255
		-1
Maximum velocity value for activation values	V_{max_a}	1
Minimum velocity value for activation values	V_{min_a}	-1
Activation threshold for activation values	T	0.5
Maximum coordinate value for positions	Z_{max}	255 (RGB)
		1 (HSL)
Minimum coordinate value for positions	Z_{min}	0
Maximum coordinate value for activation values	Z_{max_a}	1
Minimum coordinate value for activation values	Z_{min_a}	0
Size of the spatial window	$windowSize$	5
Neighborhood topology and size of the swarm	/	global, $[1, popSize - 1]$

Table 3.2: Our MEPSO implementation parameter overview.

3.2.1 Acceleration constants

Recall that in Equation 2.1 (page 4) that the velocity update rule is defined as follows:

$$\vec{V}_p(t+1) = \omega \cdot \vec{V}_p(t) + R(0, c_1) \cdot (\vec{P}_p(t) - \vec{Z}_p(t))' + R(0, c_2) \cdot (\vec{P}_{gp} - \vec{Z}_p(t))'$$

Here, c_1 and c_2 are the acceleration constants, which determine the ratio between the movement towards the particles own best position so far, and the global best position found so far. (see Section 2.2).

The authors of the paper describing MEPSO [7] did use acceleration constants, with c_1 increasing from 0.35 to 2.4, and c_2 decreasing from 2.4 to 0.35. They do not motivate

this configuration, and because they are unclear about their exact number of iterations, it is also unclear when, and with what amount, each acceleration constant is updated.

If not mentioned in the results, we ignored the acceleration constants by letting them both have the value of 1. We separately tested their choice with a similar configuration, by letting c_1 increase from 0.1 to 2, and c_2 decrease from 2 to 0.1. In each of the 20 iterations, we changed them with a value of 0.1. We also tested the reverse case, letting c_2 increase from 0.1 to 2, and c_1 decrease from 2 to 0.1.

3.3 MEPSO implementation choices

3.3.1 Spatial window

The concept of a spatial window to exploit the fact that neighboring pixels will often show a high degree of correlation has been discussed in Section 2.3.2. As pointed out there, the values for r and t in the formula for u'_{ij} , Equation 2.13, will determine the relative importance of the spatial information. In this setup, r and t will be assumed to be 1, as in Equation 2.14. These settings are also used in the original experiments with the MEPSO algorithm.

However, we will not be able to make a good comparison with regard to the spatial window. Its effect will be largely determined by its dimension relative to the dimensions of the image to be clustered, but Das et al. [7] did not mention the size of the images that their algorithm was applied on.

3.3.2 Multi-elitist strategy

The multi-elitist strategy of MEPSO, as discussed in Section 2.4.2, has been adopted unaltered.

3.3.3 Clusters with one data point

As discussed in Section 2.4.3, when an active cluster centroid of a particle contains only one or zero data points, the cluster centroids of this particle are reinitialized.

Would a cluster centroid have only one data point assigned to it, the centroid and its member could become equal, and this situation will result in a division by zero when calculating the membership degrees u_{ci} for data point i —recall Equation 2.7:

$$u_{ci} = \frac{1}{\sum_{l=1}^k \left(\frac{\|\vec{X}_i - \vec{C}_c\|}{\|\vec{X}_i - \vec{C}_l\|} \right)^{\frac{2}{m-1}}}$$

Such a case could simply be intercepted. However, cluster centroids should in principle never have less than two data points assigned to it, because that goes against the definition of a “cluster”. In Section 2.4.3 we explained that it is unclear how this reinitialization exactly happens in the original MEPSO implementation.

Here, we have adopted a simple method to deal with these erroneous clusters, that only affects the relevant cluster centroid of the particle.

Recall from Section 2.2.3 that each cluster in a particle has an activation value, and that the cluster is deactivated (not used in any calculations) if this value is below the activation threshold, T , which is 0.5.

Every iteration, after computation of the \mathbf{U} -matrix, we check if an active cluster contains less than two data points.

If the cluster contains zero data points, its corresponding activation value $a_{p,k}$ is set to 0.49. This seems reasonable, since we do not want the computation of the fitness value (the Xie-Beni index) being affected by clusters that have no data members. Recall from Section 2.2.3 that the denominator of the Xie-Beni index, partly consists of the smallest distance between two active clusters.

If the cluster contains only one data point, the cluster centroid is reset with new random values, and its corresponding activation value $a_{p,c}$ is set to 0.51. Because the cluster is assigned new random values, yet still activated, chances are high that the fitness value of the particle is negatively affected by it, because its sole member now has a high chance of not much corresponding to the cluster centroid anymore. In the next iteration, calculating the \mathbf{U} -matrix again, this cluster will either get more members, or the data member that belonged to this centroid has found a highest membership degree to another cluster, and this cluster will therefore become deactivated (having now zero data points attributed to it).

3.4 Choice of input images

An overview of the input images is given in Table 3.3.

In the paper that proposed MEPSO (Das et al. [7]), the authors show the results for only three different input images (depicted in grayscale): an image composed of three different textures, an image containing peppers with highlight and shadow, and an image of a fMRI brain scan. Indeed, it makes sense to test particularly how an image segmentation algorithm performs for both images with texture(s) on one hand, and for images with highlights and/or shadows on the other hand. Segmenting objects with lighting differences is not a trivial task for image segmentation algorithms, and segmenting (complexly) textured objects is regarded as the most difficult problem for any image segmentation algorithm [5].

The textured image we tested was *texture*, and as images with lighting effects in them, we used *pepper1* and *pepper2*. The difference between these two is that in *pepper1*, the background behind the pictured pepper is white, which means that the background has an undefined hue value. The image *pepper2* is the same image as *pepper1*, but with the background set to a purple color.

As variations of the same picture, *simpleSBdiff* and *simpleSBdiff* were tested. The composition of both images is the same. In the HSL color model, the background and the two leftmost rectangles in *simpleSBdiff* and *simpleSBdiff* have the same hue value. In *simpleBdiff* there are only lightness differences with respect to the hue: the background

and the two leftmost rectangles differ from each other in the value for their lightness component. In *simpleSBdiff* there are both lightness and saturation differences with respect to the hue: the background now differs in the value for its saturation component.

Finally, we tested an image called *flower*, which was also examined by Patrascu [15]. The image *flower* contains relatively many texture and lighting differences and therefore should be hard to segment correctly.

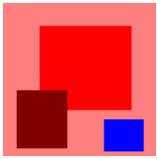
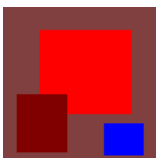




Name	Miniature	Dimension (width × height, pixels)	Nr. of distinct colors	Expected nr. of clusters	Nr. of pix- els with un- defined hue
<i>simpleBdiff</i>		100 × 100	3	4	0
<i>simpleSBdiff</i>		100 × 100	3	4	0
<i>pepper1</i>		100 × 100	2807	3	4950
<i>pepper2</i>		100 × 100	2977	3	0
<i>texture</i>		100 × 100	8227	3	0
<i>flower</i>		103 × 100	8962	3	24

Table 3.3: Overview of input images

3.5 Measures used for the results

What constitutes a “good” outcome of an image clustering algorithm? Here we use three measures. First, we use a cluster validity index, the Xie-Beni index, as outlined in Section 2.2.3. The lower this measure, the better the solution. Second, we measure the entropy of the clustering partition. Again, the lower this measure, the better the solution.

However, both of these measures do not guarantee a solution that is in accordance with the human view. For example, when clustering a very complex image with many (from the viewpoint of a human) distinct areas and over 1000 colors, we would say the solution would be faulty if it contained just two different clusters. Even more if these two clusters would be based on e.g. the amount of illumination instead of the “color” (hue) of the objects in the image. But both the Xie-Beni index and the Partition Entropy for such a solution could be very low (i.e., good).

Therefore, we also introduce a new measure, called the Cluster Difference Error. The Cluster Difference Error is the squared difference between the number of different clusters which the solution produces, and our assessment of the numbers of clusters that we expect it should come up with. For our assessments of this number per image, see Table 3.3.

This still does not rule out the possibility that the partition of the clusters might be faulty. We will therefore also show the resulting best, most typical (i.e. average) and worst resulting images of variations of the algorithm, so that these results can be compared to their original input images. We will call this the visual results.

Chapter 4

Visual results

We will first look at the results using our MEPSO implementation with different color representations. Then we will look at the results that the Soft k-means algorithm yields. Finally we will take a look how different parameter settings affect the performance of our MEPSO implementation, especially with regard to the acceleration constants.

The images we tested are shown in Table 3.3 (page 26). We will show the best, most typical (i.e. average) and worst resulting images that the algorithm came up with for each condition-image combination. For each condition-image combination, the algorithm was run at least 20 times.

What constitutes the quality of a resulting image is determined subjectively here, because no exact measures exist for this purpose (but for some measures on the resulting cluster partitions, see Chapter 5). We have not taken into account the similarity of the colors *an sich*, as this thesis is about segmenting only. We estimated the correctness of the segmentation with regards as to how humans would segment the distinct objects on the pictures. This includes not producing separate segments for shadows, highlights, and texture differences in the resulting images. The amount of clusters that one should expect to be found for each image is defined by us in Table 3.3 (page 26).

Because we have used relatively simple, straightforward images, containing either strict boundaries or simple objects familiar to humans, we do not expect our lack of strict measures for judging the visual results to have a significant bias.

When the resulting image does not have any segments (i.e. the algorithm ended with only one cluster active), the outcome is denoted as a “failure”.

4.1 Comparison with original MEPSO

In the paper outlining the MEPSO algorithm, the authors showed only one result for each image, without information if this was the best, average, or worst result. We assume they showed their best results, so we also show our best results.

Their input images were in grayscale. We used a colored pepper image and texture image highly similar to their pepper image and texture image, the same color representation (RGB), and an algorithm as similar as possible. We achieved comparable or

better results in these best cases.

4.1.1 Pepper image

Our image is of a simpler composition, but the goal here was to examine the segmentation of an image with lighting differences (the highlights and shadows on the pepper(s)). We can see their implementation produced an image with different segments for the highlights and shadows —each pepper in their resulting image is represented by more than one segment. We can see that our implementation has produced a result in which the shadows on the pepper are not regarded as needing to belong to a separate segment, and a large part of the pixels constituting the main highlight on the pepper is also correctly recognized as belonging to the same cluster as the pepper itself.

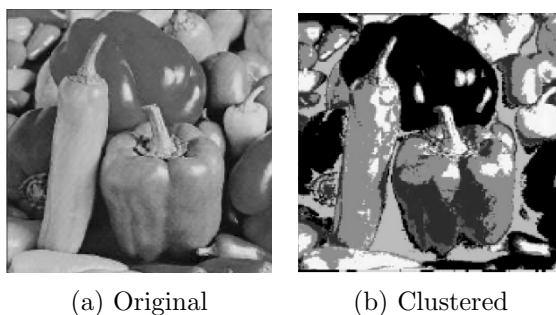


Figure 4.1: Original MEPSO (RGB color representation) (taken from [7])

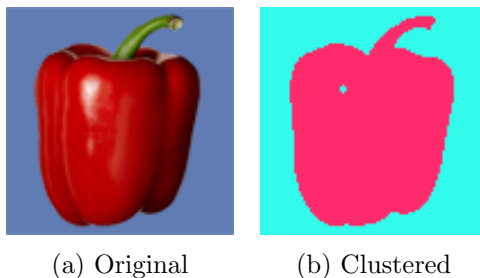


Figure 4.2: Our MEPSO (RGB color representation)

4.1.2 Texture image

Here, the goal was to examine how the algorithms would compare for segmenting images with texture(s). Both input images are comparable and the resulting images are also comparable. Just like the original MEPSO algorithm, our algorithm correctly identifies the three segments.

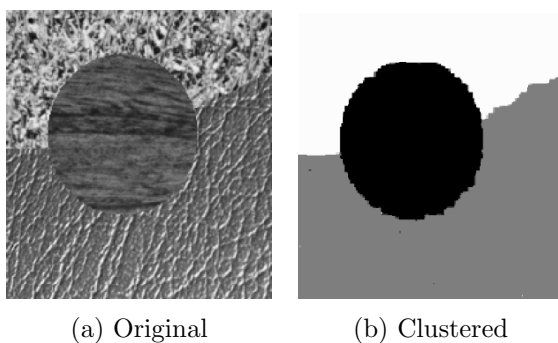


Figure 4.3: Original MEPSO (RGB color representation) (taken from [7])

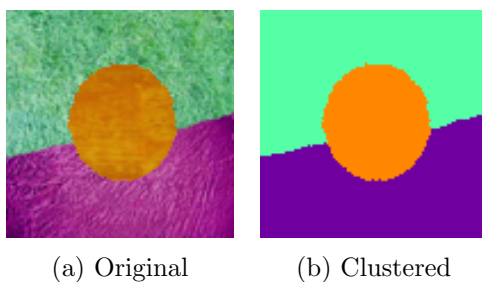


Figure 4.4: Our MEPSO (RGB color representation)

4.2 Color representations

4.2.1 Pepper images

Using the RGB color representation (Figure 4.5), the highlight on the pepper was still present in all of the resulting images. Using the HSL with D_{LSH} color configuration (Figure 4.6), producing a resulting image for *pepper1* or *pepper2* often failed, albeit not in the majority of cases. Also, the worst resulting images in this configuration tend to be even more fragmented (i.e. more segmented) than the original picture.

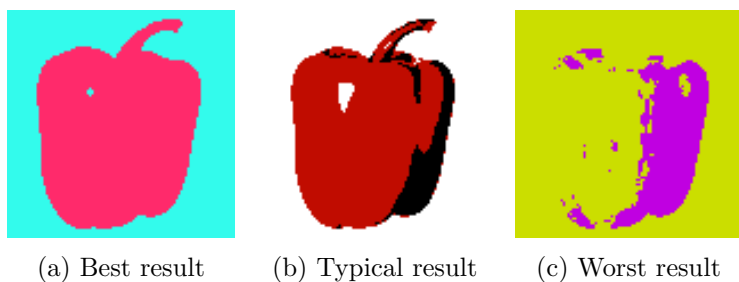


Figure 4.5: RGB color representation

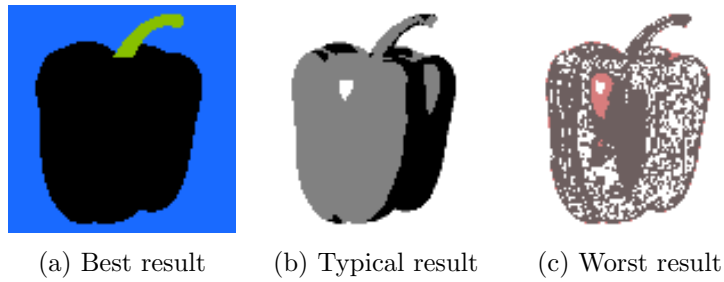


Figure 4.6: HSL color representation using the D_{LSH} distance measure

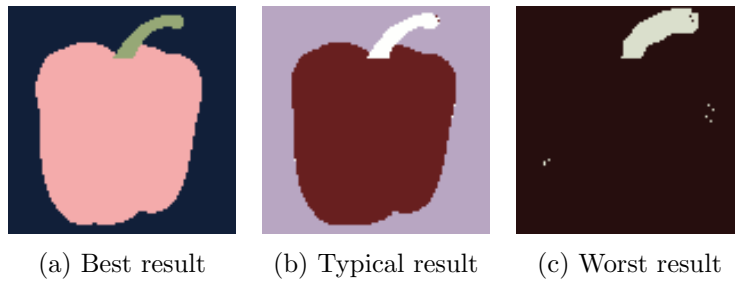


Figure 4.7: HSL color representation using the D_H distance measure

4.2.2 SimpleBdiff image

In the RGB representation (Figure 4.8), the algorithm has a tendency to cluster the left and middle rectangle together, because they have the smallest distance in the RGB cube (one half on the R scale).

In the HSL with D_{LSH} condition (Figure 4.9), most of the times the algorithm failed, i.e. produced an image with no segments at all. This probably happens because there are no saturation (i.e. chromaticity) differences in this picture. This is further elaborated on in Chapter 6.

The HSL with D_H condition (Figure 4.10) produces results as predicted: only the components with different hue are segmented.

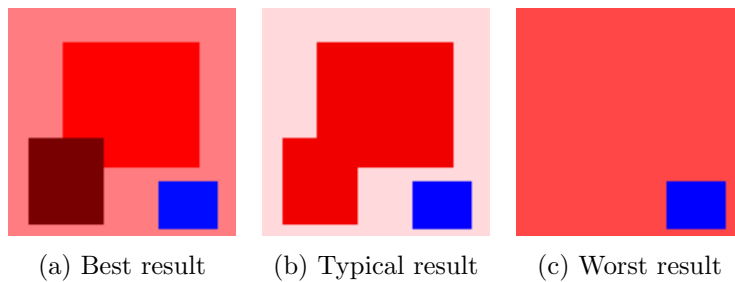


Figure 4.8: RGB color representation

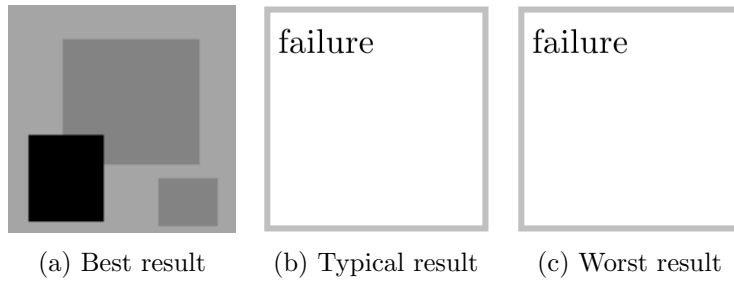


Figure 4.9: HSL color representation using the D_{LSH} distance measure

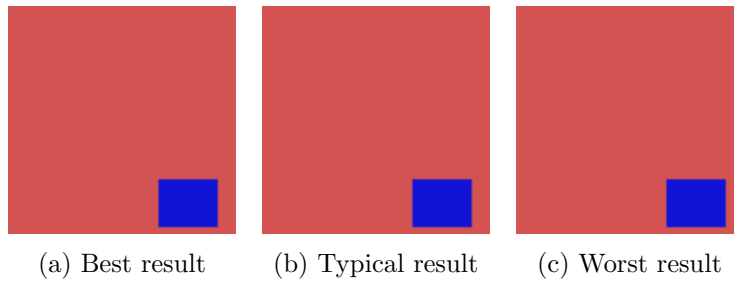


Figure 4.10: HSL color representation using the D_H distance measure

4.2.3 SimpleSBdiff image

The RGB color representation condition is not shown here —it yielded the same kind of results as with the *simpleBdiff* picture (Figure 4.8).

Although in this HSL with D_{LSH} condition (Figure 4.11) the input image has hue, saturation as well as lighting differences, the algorithm still most of the times fails to produce a segmented image like one would perhaps expect. This is further elaborated on in Chapter 6.

The HSL with D_H condition (Figure 4.12) again produces results as predicted: only the components with different hue are segmented.

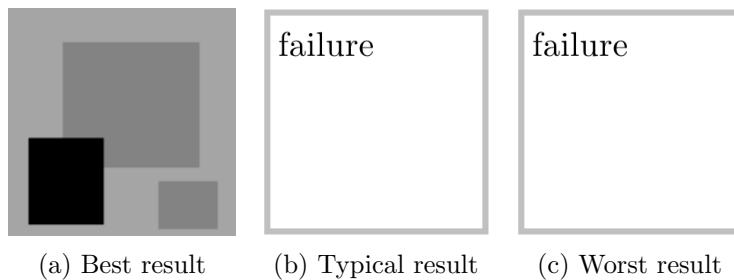


Figure 4.11: HSL color representation using the D_{LSH} distance measure

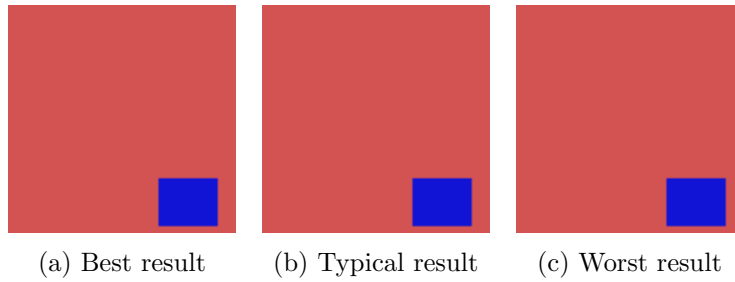


Figure 4.12: HSL color representation using the D_H distance measure

4.2.4 Texture image

In the RGB color representation condition (Figure 4.13), typically one of the three “intuitive” or expected segments in the *texture* image was distorted. That is, two of the three expected segments were correctly discerned, but the other one was fragmented, its pixels assigned to multiple clusters.

In the case of HSL with D_{LSH} (Figure 4.14), producing a resulting image for *texture* failed or produced nonsense half of the time, but produced sensible results the other half of the time.

In the HSL with D_H condition (Figure 4.15), the best and typical results are much like in the HSL with D_{LSH} condition —but the algorithm less often fails, and the worst outcomes are not as bad.

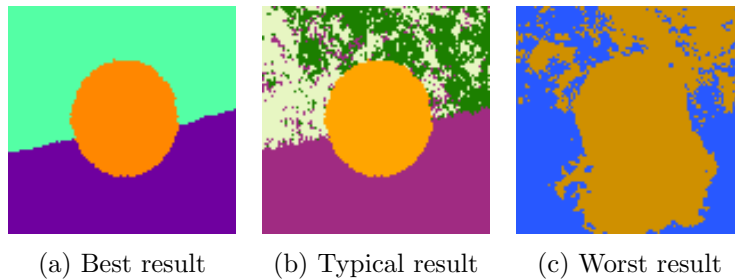


Figure 4.13: RGB color representation

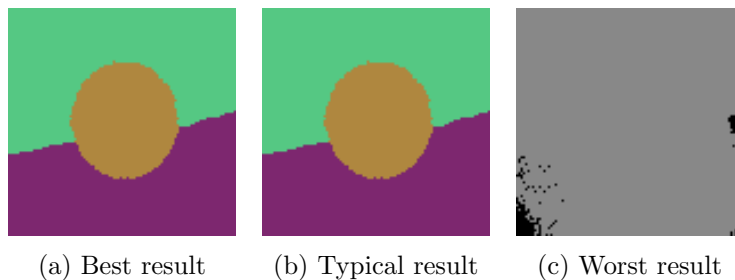


Figure 4.14: HSL color representation using the D_{LSH} distance measure

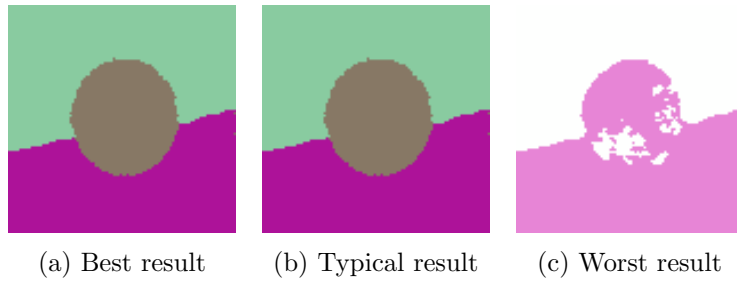


Figure 4.15: HSL color representation using the D_H distance measure

4.2.5 Flower image

Perhaps surprisingly, considering the previous results, the HSL with D_{LSH} condition (Figure 4.17) yielded superior results for the *flower* image in the best case. This is further elaborated on in Chapter 6.

In the best case of HSL with D_H (Figure 4.18), we can still see some pixels in the background of the image are attributed to the same cluster as the “flower”.

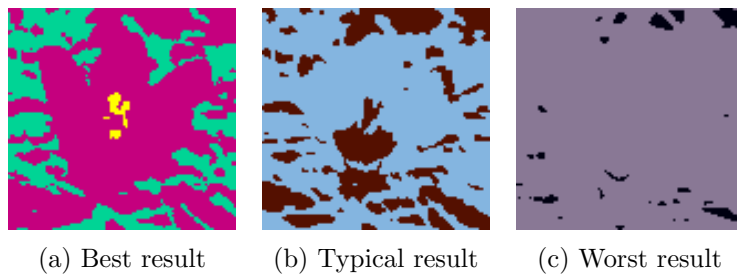


Figure 4.16: RGB color representation

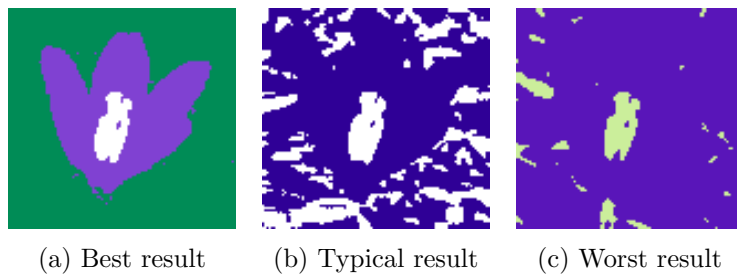


Figure 4.17: HSL color representation using the D_{LSH} distance measure

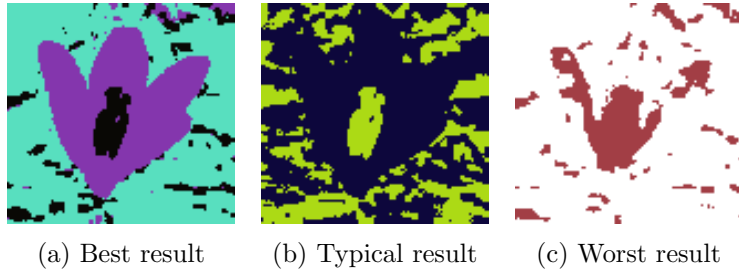


Figure 4.18: HSL color representation using the D_H distance measure

4.3 Comparison with Soft k-means

To examine what exactly the advantages or disadvantages of the PSO clustering approach can be, we will compare our MEPSO implementation to the standard Soft k-means (SKM) clustering algorithm (as described in Section 2.2.2). The amount of clusters was set to 10 and the number of iterations to 20. Recall that when testing MEPSO, we set the maximum amount of clusters to 10, the number of iterations to 20 and the number of particles (population size) to 10. We did not use the method to reset clusters with less than one data point (see Section 3.3.1), because that method makes use of the activation value of a cluster centroid, and the cluster centroids do not have separate activation values in SKM. The multi-elistist strategy (see Section 2.4.2) could also not be applied, because that strategy needs a set of candidate solutions and SKM works with just one (candidate) solution.

When using Soft k-means, using the standard method to update the cluster centroids as defined in Equation 2.6 (page 7):

$$\vec{C}_c \leftarrow \frac{\sum_{i=1}^n u_{ci}^m \cdot \vec{X}_i}{\sum_{i=1}^n u_{ci}^m}$$

we have to ask ourselves what to do if the hue component of one data point is undefined. Theoretically, UNDEFINED added to any number results in UNDEFINED. This means that the hue components of all newly calculated centroids will become undefined if one data point has an undefined hue value. To work around this, we simply treated the value of the hue component as zero if it was undefined for each data point \vec{X}_j in the above equation.

Soft k-means generally performs poorly compared to MEPSO. The main culprit for this is the fact that it has to use a fixed amount of (all active) cluster centroids.

4.3.1 Pepper images

The results using the RGB color representation (Figure 4.19) and the HSL with D_{LSH} representation (Figure 4.20) are worse than when using our MEPSO implementation.

They are much too detailed. This is probably due to the fact the correct amount of clusters is a priori set to 10.

The best result seems to be achieved by using the HSL with D_H color representation (Figure 4.21). At first glance this result seems just as good as when using MEPSO, but upon closer examination, there are much more clusters defined than when using MEPSO. These “redundant” clusters have their members in lonely pixels at the “stem” of the “pepper”.

Note, though, that the results using SKM are more consistent, with the worst result being very similar to the best result closely.

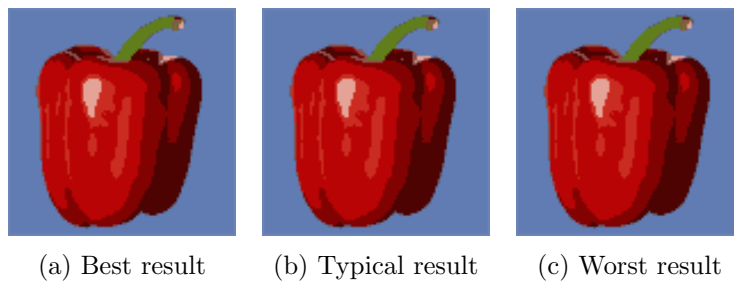


Figure 4.19: RGB color representation

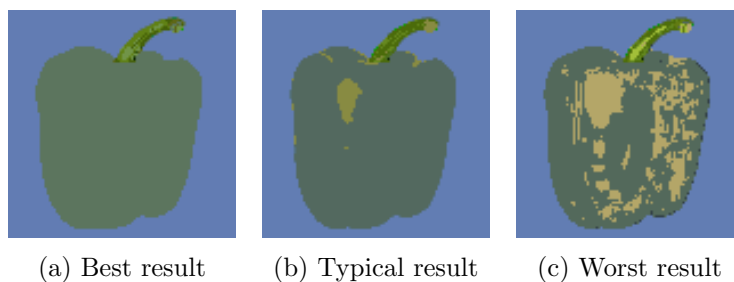


Figure 4.20: HSL color representation using the D_{LSH} distance measure

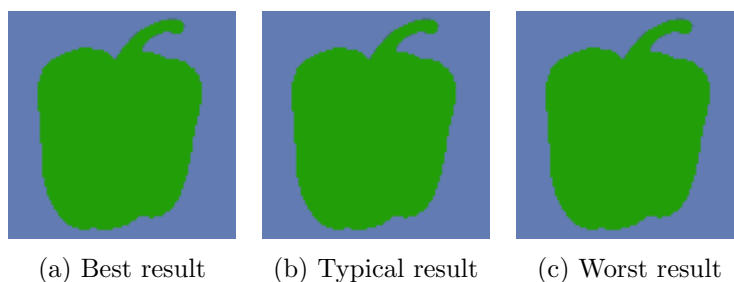


Figure 4.21: HSL color representation using the D_H distance measure

4.3.2 SimpleBdiff image

Whichever color representation was used, the algorithm failed most of the time to produce a clustered image. This is probably due to the fact that the correct amount of clusters is a priori set to 10. This is further discussed in Chapter 6.

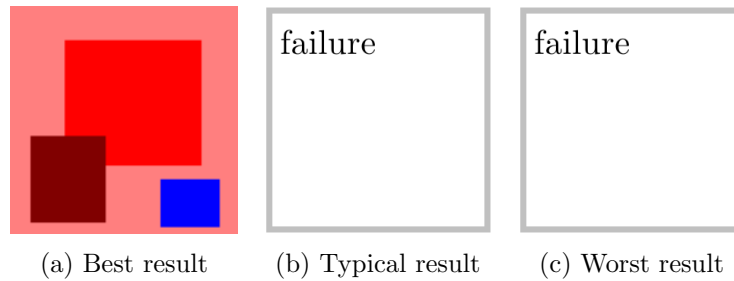


Figure 4.22: RGB color representation

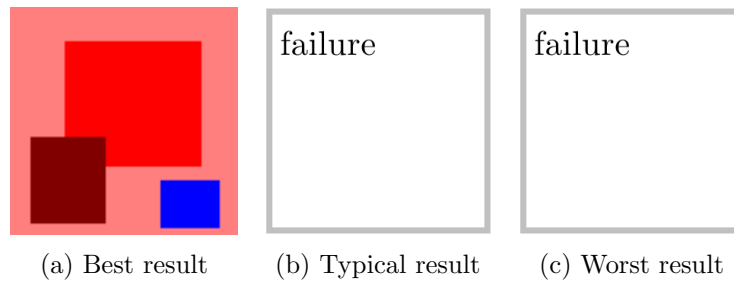


Figure 4.23: HSL color representation using the D_{LSH} distance measure

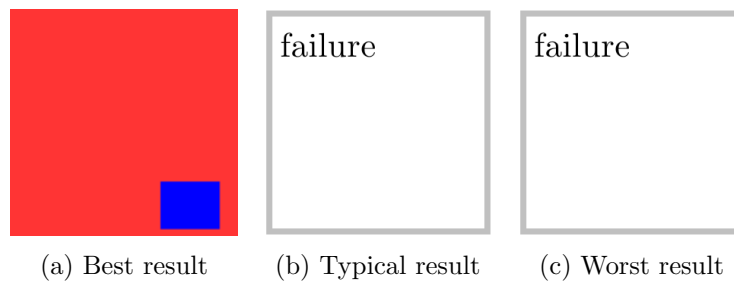


Figure 4.24: HSL color representation using the D_H distance measure

4.3.3 SimpleSBdiff image

The results are similar to those for the *simpleBdiff* image. Whichever color representation was used, the algorithm failed most of the time to produce a clustered image.

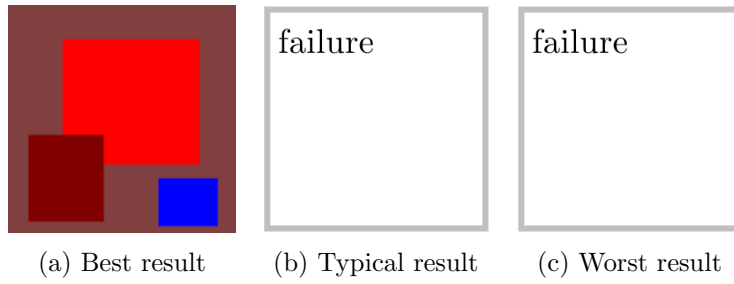


Figure 4.25: HSL color representation using the D_{LSH} distance measure

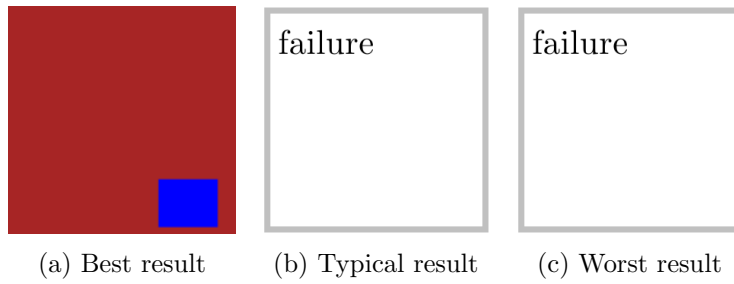


Figure 4.26: HSL color representation using the D_H distance measure

4.3.4 Texture image

Whichever color representation used, the algorithm never failed on the *texture* image. We assume this does not happen because this image can easily be segmented in 10 clusters (because of the texture). In most cases, the algorithm made separate clusters of only one of the three (to humans) identifiable segments.

The HSL with D_{LSH} color representation (Figure 4.28) yielded the best best result for this image, it made separate segments of two of the three expected segments.

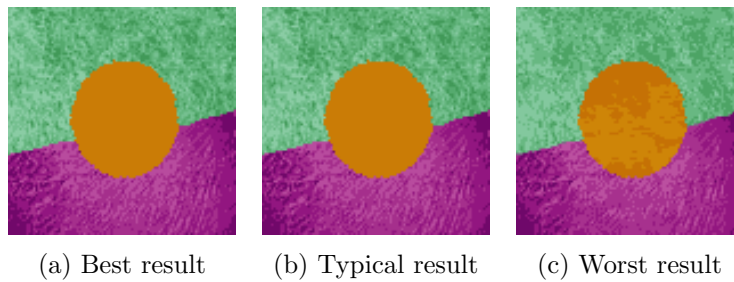


Figure 4.27: RGB color representation

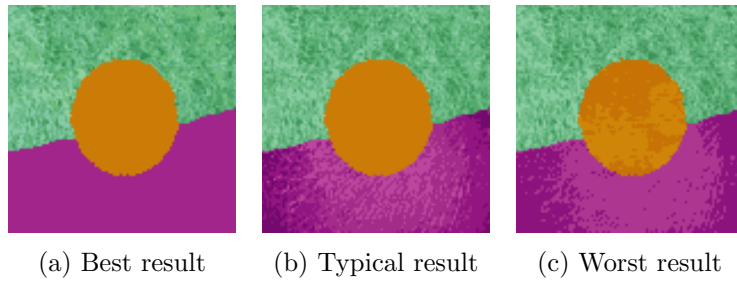


Figure 4.28: HSL color representation using the D_{LSH} distance measure

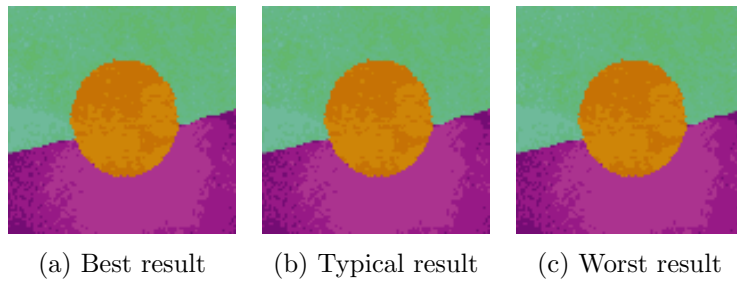


Figure 4.29: HSL color representation using the D_H distance measure

4.3.5 Flower image

As with the *pepper* and *texture* image, the results using the RGB color representation (Figure 4.30) are far too detailed.

All results using the HSL with D_{LSH} color representation (Figure 4.31) are also far too detailed. Using the HSL with D_{LSH} color representation, the typical and worst results seem to be in grayscale, but in fact they are of a blueish color, all pixels having a very low (but non-zero) saturation value. This unexpected effect is further discussed in Chapter 6.

The best and typical results using the HSL with D_H color representation (Figure 4.32) are quite nice, if ones goal would be to solely segment the “flower” area in the image, regardless of what would happen to the background. The background in this case is still rendered as highly detailed, consisting of many segments.

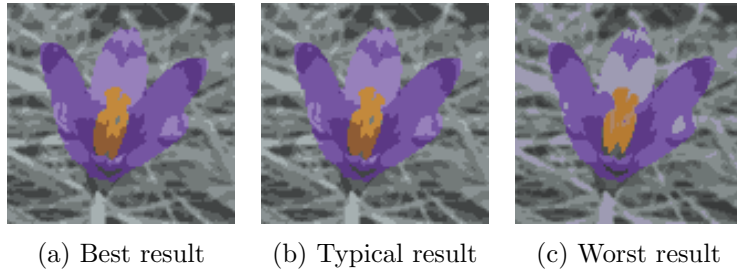


Figure 4.30: RGB color representation

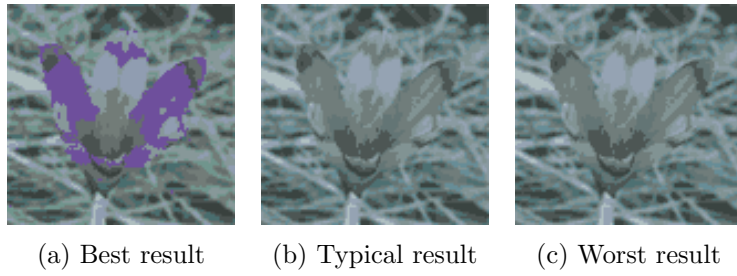


Figure 4.31: HSL color representation using the D_{LSH} distance measure

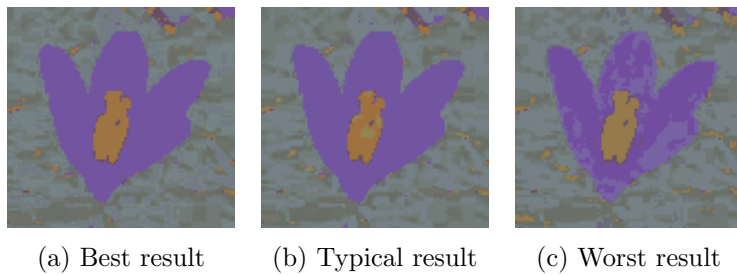


Figure 4.32: HSL color representation using the D_{H} distance measure

4.4 Parameters

4.4.1 Acceleration constants

To examine the different settings for the acceleration constants, we only used the color representation HSL with D_H , on the three images *pepper2*, *texture* and *flower*.

We did not use the images *simpleBdiff* and *simpleSBdiff* because they were specifically designed to test different color representations and the effect of assuming too many cluster centroids as we did in the previous sections. Because of their simple composition it is assumed the acceleration constant settings should not affect the clustering solutions for them much compared to those in Section 4.2. The image *pepper1* was also not tested because it only differs with respect to *pepper2* in a color-relevant way. The color representation used should not affect the acceleration constants settings at all, so we simply used the color representation HSL with D_H , which seemed to yield on average the best results in Section 4.2.

For all three images, using the configuration of c_1 decreasing, c_2 increasing produced on average better resulting images than using the configuration of c_1 increasing, c_2 decreasing (the method used in the original MEPSO implementation). Not only produces the configuration of c_1 decreasing, c_2 increasing better results than when using the configuration c_1 increasing, c_2 decreasing, it also produces on average better results than when using no acceleration constants configuration at all ($c_1 = c_2 = 1$). This advantage of “ c_1 decreasing, c_2 increasing” is further discussed in Chapter 6.

Pepper images

The configuration of c_1 decreasing, c_2 increasing (Figure 4.34) produced on average better resulting images than using the configuration of c_1 increasing, c_2 decreasing (Figure 4.33). The best results are comparable, but the typical and worst results were better using the configuration of c_1 decreasing, c_2 increasing.

Comparing the configuration of c_1 decreasing, c_2 with the configuration of no acceleration constants configuration at all ($c_1 = c_2 = 1$) in Section 4.2 (Figure 4.7), the best and typical results are comparable, but the worst result is better

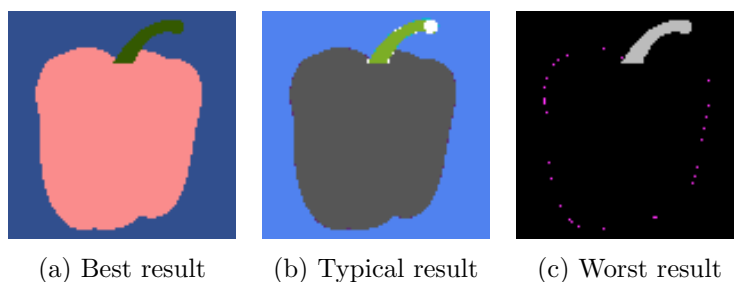


Figure 4.33: c_1 increasing, c_2 decreasing

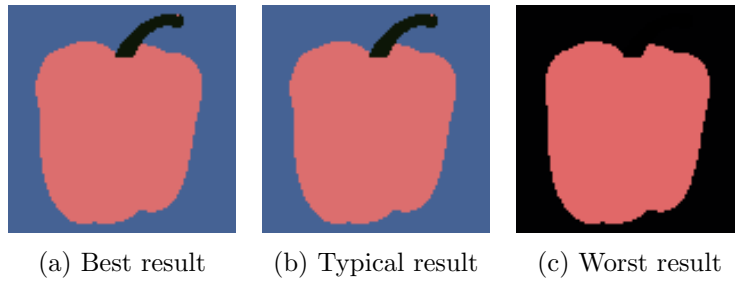


Figure 4.34: c_2 increasing, c_1 decreasing

Texture image

The configuration of c_1 decreasing, c_2 increasing (Figure 4.36) produced on average better resulting images than using the configuration of c_1 increasing, c_2 decreasing (Figure 4.35). The best results are comparable, but the typical and worst results were better using the configuration of c_1 decreasing, c_2 increasing.

Comparing the configuration of c_1 decreasing, c_2 with the configuration of no acceleration constants configuration at all ($c_1 = c_2 = 1$) in Section 4.2 (Figure 4.15), the best and typical results are comparable, but the worst result is better.

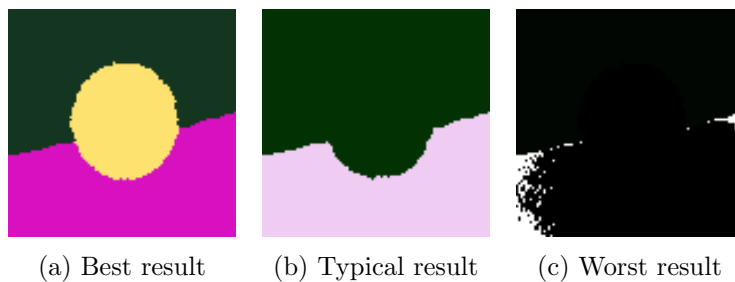


Figure 4.35: c_1 increasing, c_2 decreasing

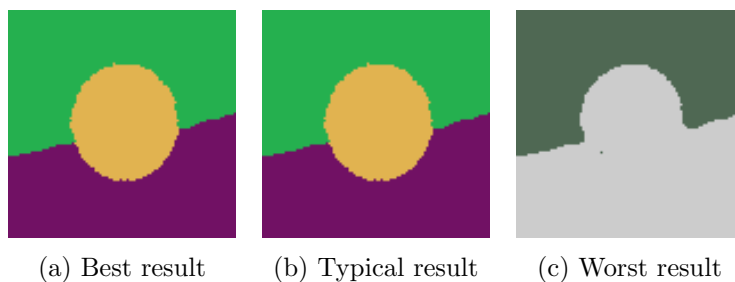


Figure 4.36: c_2 increasing, c_1 decreasing

Flower image

The best result is slightly better in the condition of c_1 increasing, c_2 decreasing (Figure 4.37), and the worst result is comparable, but the typical result was better using the configuration of c_1 decreasing, c_2 increasing (Figure 4.38).

Comparing the configuration of c_1 decreasing, c_2 with the configuration of no acceleration constants configuration at all ($c_1 = c_2 = 1$) in Section 4.2 (Figure 4.18), the best and worst results are comparable, but the typical result is better.

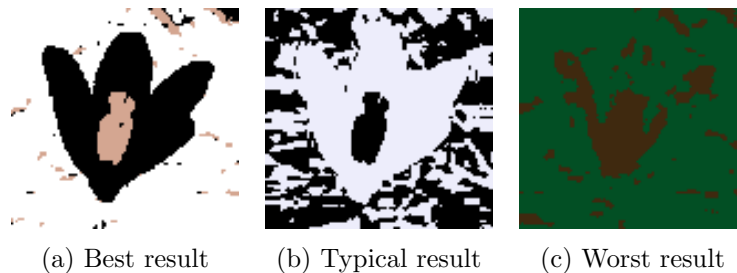


Figure 4.37: c_1 increasing, c_2 decreasing

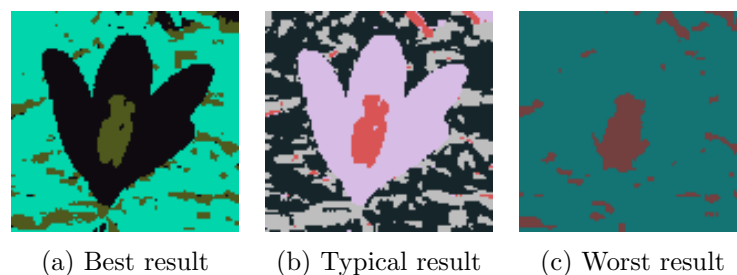


Figure 4.38: c_2 increasing, c_1 decreasing

4.4.2 Spatial window

In conducting our experiments so far we had an equal number of runs for each image-condition combination in which the spatial window method was turned on or off. Serendipitously, we noted a peculiar effect occurring with regard to the images than *pepper1* and *pepper2*.

Using MEPSO and the color representation HSL and D_H , clustering *pepper1* in the condition of using the spatial window, led to an increase in the space occupied by the pepper in the picture, that originally was white background (see Figure 4.39). Another notable finding was that when using Soft K-means algorithm and the HSL with D_{LSH} color representation, if the spatial window method was turned off, the algorithm produced notably worse pictures compared to when the spatial window method was turned on. They were all almost exactly the same as the one in Figure 4.40. See Section 4.3.1 for the typical result when using the spatial function.

In all other cases (using MEPSO and one of the two other color representations), for *pepper2*, turning the spatial window method on did not produce distinguishably different resulting images.



Figure 4.39: Typical result for image *pepper1* in the HSL and D_H condition, using the spatial window.



Figure 4.40: Typical result for *pepper2* using SKM with the HSL with D_{LSH} color representation, without using the spatial window.

Chapter 5

Data results

In this Chapter we present the statistical results for the validity measures (the Xie-Beni index, the Partition Entropy and the Cluster Difference Error —see Section 3.5) for each condition. For obtaining these measures, we only used the three images *pepper2*, *texture* and *flower*, unless stated otherwise. The tables show for each validity index the mean and, in brackets, the standard deviation. For each validity index, a lower result means a better result.

For each condition-image combination, the algorithm was run at least 20 times. For readability purposes the measures of the Partition Entropy and Cluster Difference Error are rescaled to fit in $[0, 1]$. The Xie-Beni index is theoretically in $[0, \infty]$, but in our results, its upper limit is in most cases also 1.

Solutions with no clustering at all —meaning, only one cluster was active in the resulting “partition” —are denoted as “failures”. We excluded these so-called failures in these results. Instead we denoted the percentage of failures. The reason why we exclude the failures when calculating the means and standard deviation is, first, because of the way the Xie-Beni index is computed in our code, its result will still be relatively small (i.e. good) for failures, and second, the Partition Entropy will always be the lowest for failed solutions (because the partition is fully crisp in that case).

5.1 Comparison with original MEPSO

Here our implementation of MEPSO used the color representation RGB, just the original MEPSO implementation of Das et al. [7] did. The Partition Entropy results as displayed in Das et al. [7], are rescaled to fit in $[0, 1]$, to match the domain to which we scaled our Partition Entropy results.

In the case of the *texture* image, 15% of the results were failures. In the case of the *pepper2* image, 5% of the results were failures.

Note that their standard deviations are much lower, and their entropy values are much higher, in every case. This could be the case because they used a population that was twice as large as ours (40 particles versus 20 particles), and/or because they used

different settings for the acceleration constants (c_1 increasing, c_2 decreasing in their case, $c_1 = c_2 = 1$ in our case).

For the texture image, our implementation yields better results on both validity indices. In this case, the Xie-Beni index differs relatively much between the two algorithms. For the lighting effects image, our implementation performs worse on the Xie-Beni index. But in this case the difference between the average Xie-Beni index values of the two algorithms is relatively small.

Image	Validity index	Algorithm	
		Original MEPSO	Our MEPSO
Texture image (<i>texture</i>)	Xie-Beni	0.7283 (0.0001)	.1058 (.023)
	Partition Entropy	0.801673 (0.000054)	.0786 (.054)
Lighting effects image (<i>pepper2</i>)	Xie-Beni	0.05612 (0.0092)	.0764 (.042)
	Partition Entropy	0.267074 (0.004124)	.0555 (.073)

Table 5.1: Comparison with original MEPSO (mean and standard deviation).

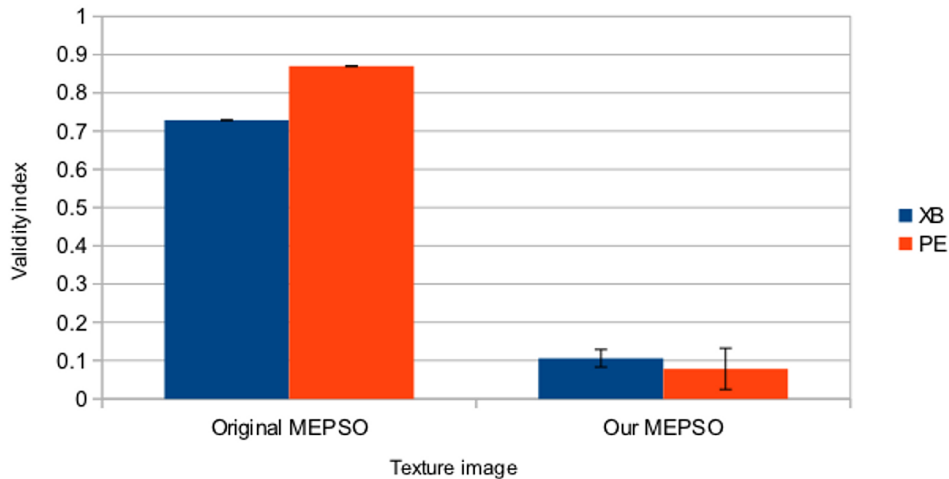


Figure 5.1: Texture image comparison with original MEPSO (Y error bar denotes the standard deviation).

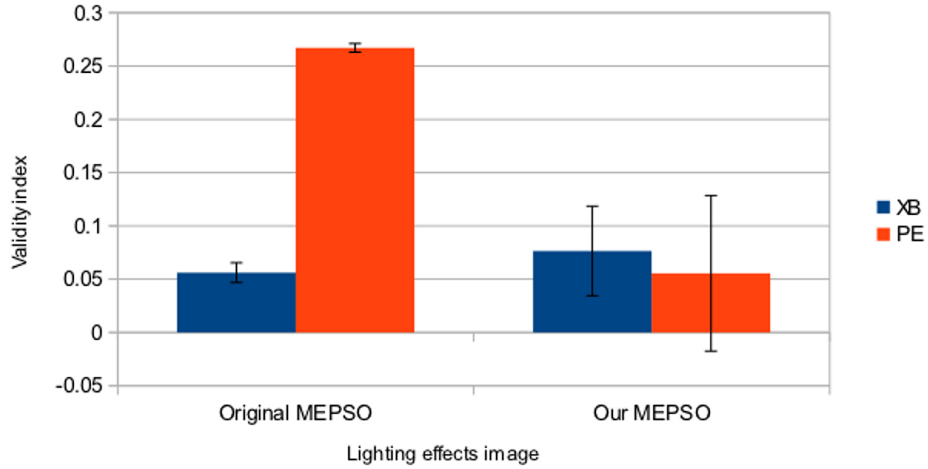


Figure 5.2: Lighting effects image comparison with original MEPSO (Y error bar denotes the standard deviation).

5.2 Color representations

In each color representation produces the lowest (i.e. best) result for a different validity index. The RGB representation produces the worst result for two of the three indices. The HSL with D_{LSH} representation does not produce the worst result for any of the three indices.

This is not consistent with what we found earlier in see Section 4.2. There we found that the HSL with D_H color representation, which here only produces the best result for the Partition Entropy measure, produced the best resulting images. However, the HSL with D_H color representation does have the lowest percentage of failures.

Validity index	Color representation		
	RGB	HSL and D_{LSH}	HSL and D_H
Xie-Beni	.0953 (.031)	.0148 (.023)	.0267 (.027)
Partition Entropy	.0682 (.059)	.0212 (.024)	.0131 (.018)
Cluster Difference Error	.0127 (.017)	.0154 (.021)	.0299 (.077)
Percentage of (excluded) failures	15	22	0

Table 5.2: Comparison between different color representations (mean and standard deviation).

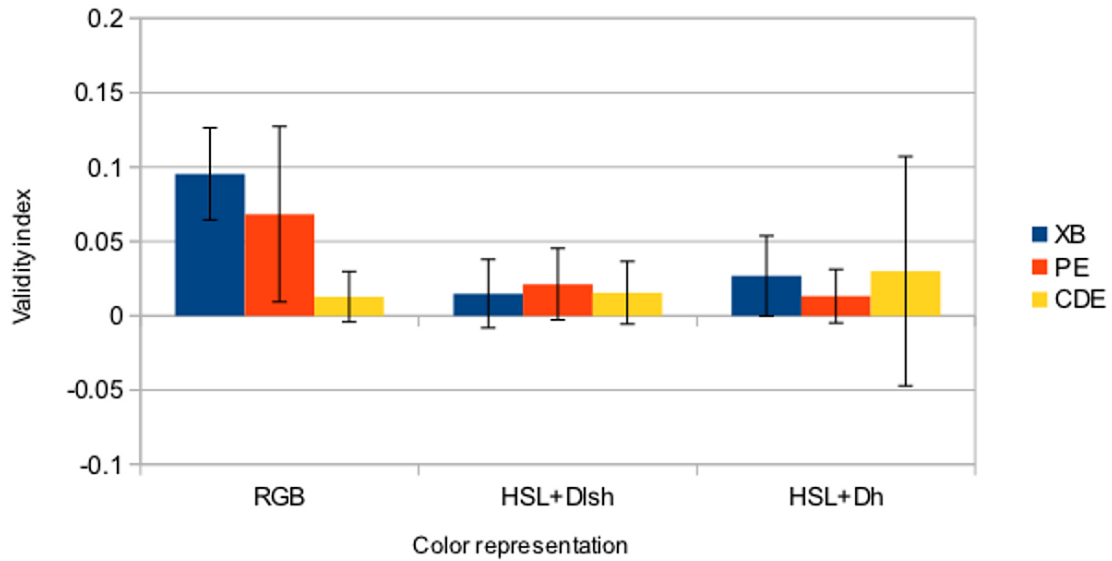


Figure 5.3: Comparison between different color representations (Y error bar denotes the standard deviation).

5.3 Comparison to Soft k-means

Here, the percentage of (excluded) failures is not relevant, because SKM in this case always has 10 active clusters.

Combining all three color representations for the three images, the result of every validity index is definitely better in the case of the MEPSO algorithm. This is in line with what we found in Section 4.4 when looking at the resulting images.

Validity index	Algorithm	
	SKM	MEPSO
Xie-Beni	1.9132E22 (1.353E23)	.0487 (.046)
Partition Entropy	.0947 (.097)	.0369 (.047)
Cluster Difference Error	.6501 (.244)	.0183 (.044)

Table 5.3: Comparison between different algorithms (mean and standard deviation).

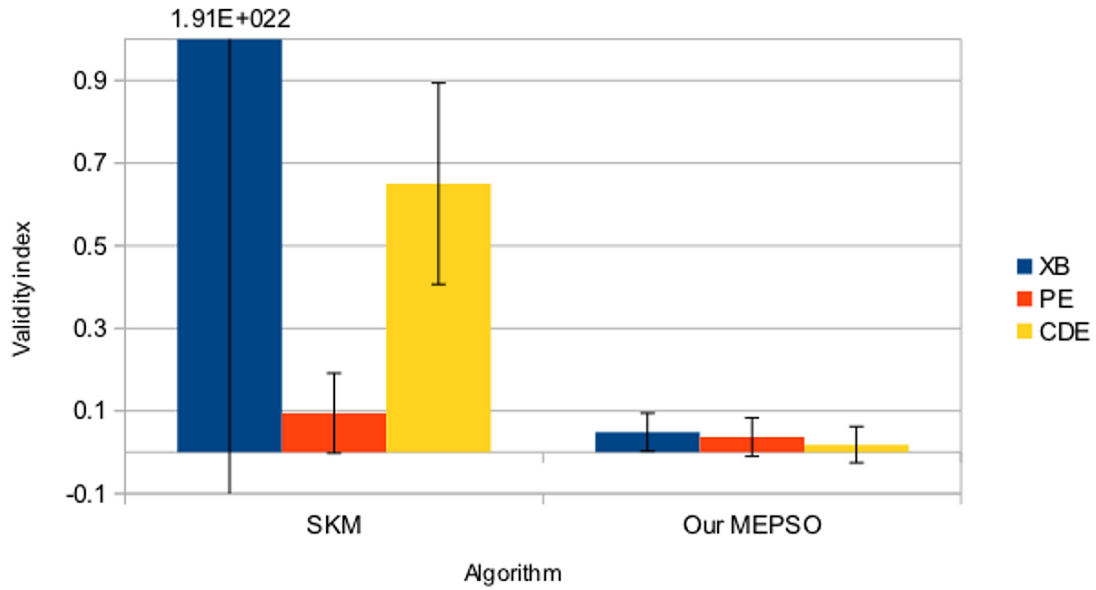


Figure 5.4: Comparison between different algorithms (Y error bar denotes the standard deviation).

5.4 Acceleration constants

Note that here only the color representation HSL with D_H was used, and only three of the five input images were used (see Section 4.4.1 for motivation).

The setting “ c_1 increasing, c_2 decreasing” produces the worst results for two of the three validity indices.

In Section 4.4.1, we found that the “ c_1 decreasing, c_2 increasing” setting produced the best images. The data for the validity indices seem to conflict that. Here we see that the setting $c_1 = c_2 = 1$ yields better results for both the Xie-Beni index and the Partition Entropy measure. However, the differences between these results and those in the case of “ c_1 decreasing, c_2 increasing”, are relatively small.

Validity index	Acceleration constants		
	$c_1 = c_2 = 1$	c_1 increasing, c_2 decreasing	c_1 decreasing, c_2 increasing
Xie-Beni	.0267 (.027)	.0313 (.026)	.0287 (.032)
Partition Entropy	.0131 (.018)	.0133 (.016)	.0132 (.019)
Cluster Difference Error	.0299 (.077)	.0268 (.066)	.0229 (.049)
Percentage of (excluded) failures	0	0	0

Table 5.4: Comparison between different acceleration constants settings (mean and standard deviation).

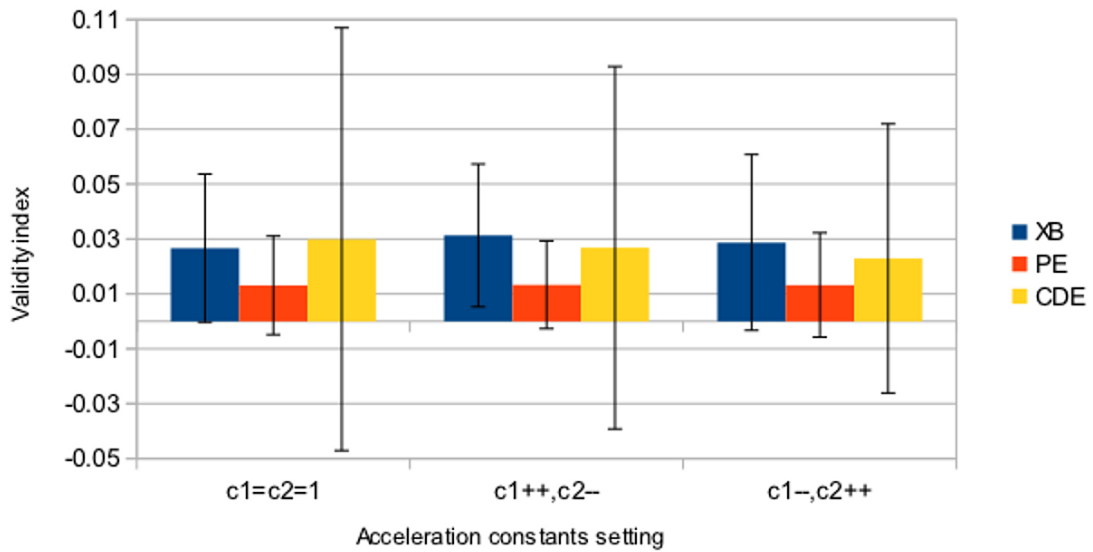


Figure 5.5: Comparison between different acceleration constants settings (Y error bar denotes the standard deviation).

Chapter 6

Discussion

Here we will briefly discuss the results we obtained and showed in Chapter 4 and Chapter 5. The conclusions of our experiments are presented in the next Chapter.

6.1 Comparison with original MEPSO

An important difference between the results of [7] and our results is that they used grayscale images and we used color images. Furthermore the authors of the original MEPSO algorithm showed only one result for each image, without information if these were their best, average, or worst results. We compared their images with our best resulting images. We used a *pepper* image and *texture* image that were in terms of lighting and texture effects highly similar to their pepper image and texture image. We did not get any strange or unexpected results.

6.2 Color representations

6.2.1 The D_H distance measure

It should be noted that clustering *pepper1* in the condition of using the spatial window using the color representation HSL with D_H , leads to an increase in the space occupied by the pepper in the picture (see Figure 4.39 on page 44) This happens because the background in the picture is fully white, which means its hue component in the HSL representation is undefined. According to the D_H difference measure used, the distance between an undefined hue value and a defined hue value is zero (see Equation 3.2 on page 20).

When the spatial window functionality is turned off, this problem tends to disappear. But note that we used a limited number of iterations. Eventually, as the number of iterations approaches infinity, the segments of the pepper and its stem would fully merge with the area that is white in the original image of *pepper*—either the white area will get the color of the pepper and stem, or the pepper and stem will become white. The reason this does not immediately happen when the spatial window is turned off, is probably

because the no-hue pixels can be assigned to any cluster centroid using the D_H distance measure, and since the maximal number of clusters was set to 10, chances are high that they are assigned to a different cluster (having any value for the hue component) than the hue-pixels early on. Fitness-wise, there is no penalty for this, and the active existence of this cluster centroid might provide benefit if this cluster centroid provides for the largest inter-cluster distance (see Equation 2.10 on page 9).

6.2.2 The D_{Lsh} distance measure

SimpleBdiff image

For the *simpleBdiff* image, in the HSL with D_{Lsh} condition, most of the times the algorithm failed, i.e. produced an image with no segments at all. This probably happens because there are no saturation (i.e. chromaticity) differences in this picture. All pixels have the maximum value for their saturation component. Now, according to the D_{Lsh} distance measure we used, based on Patrascu [15] (see Section 3.1.1), when the saturation value is maximal, the hue difference between the two pixels should be maximally taken into account, and the lightness difference minimally. Therefore, all lightness differences between pixels are disregarded. This would mean only the rightmost square should be discerned, because only the rightmost square comprises pixels with a distinct hue component, compared to the other pixels. However, even if the algorithm succeeds, this does not happen most of the times. Often other rectangles are (also) discerned. A reason for this may be that, when trying to attribute a pixel to a cluster centroid, only the two hue components are compared, but when comparing cluster centroids between each other (when calculating the XB-index), almost all of the times also the lightness components will be compared (because there is no need for the centroid to have a certain saturation value). Because only the hue component is taken into account, when wanting to decrease the distance to a pixel, simply lowering the saturation value of the cluster centroid is just as effective as altering the hue value of a the cluster centroid. When the value of saturation component is in the bottom half, this will result in a better Xie-Beni index if the largest lightness difference between two cluster centroids is just as large, or larger, than the largest hue difference between two cluster centroids. This may in rare cases result in a cluster distribution based solely on the lightness component of the original image.

simpleSBdiff image

For the *simpleSBdiff* image, algorithm still most of the times fails to produce a segmented image like one would perhaps expect.

As with the *simpleBdiff* image, only the rightmost square comprises pixels with a distinct hue component. The three front rectangles in the image have no saturation differences between them, moreover, these all have a maximally valued saturation component. However, contrary to the *simpleBdiff* image, the background has a distinctly (lower) valued saturation component. The reason the algorithm (if the algorithm suc-

ceeds) more often solely discerns the rightmost rectangle in this case (compared to when given as input *simpleBdiff*), might be because the background differs in saturation, lightness as well as hue from the rightmost rectangle. The rightmost rectangle can again be associated to a centroid with any lightness value because its saturation is maximal and only its hue component will be used to measure its distance. But because the background uses both the hue difference and lightness difference (weighted by the saturation components), instead of only the hue difference, to compute the distance, on average, the background will have a larger distance towards any cluster centroid the leftmost rectangle is associated with. And because the other squares have the same hue as the background, they can now be attributed to the same cluster as the background.

Flower image

The HSL with D_{LSH} condition produced the best solutions for the *flower* image. This might be the case because the saturation values for the pixels that comprise the “flower” in this image, are relatively high, ensuring more emphasis is put on their hue differences, while the saturation of the background is relatively low, ensuring more emphasis is put on their lighting differences. Because the lighting differences are fairly evenly distributed and not much pronounced across the image, but the “flower” has very clear hue distinctions, this probably helps rendering the *flower* apart from the background.

6.3 Comparison to Soft k-means

6.3.1 Number of clusters

SLM suffers mainly from the fact it has an a priori amount of clusters. We set that number to 10, because we set the number of default clusters in SKM also to 10.

For the images *simpleBdiff* and *simpleSBdiff*, whichever color representation was used, the algorithm failed most of the times in producing a clustered image. This is probably due to the fact that the correct amount of clusters is a priori set to 10. With such a simple image, this may result in cluster centroids becoming very similar to each other. Therefore the distribution may become unstable: pixels may have their maximal membership degree linked to a different cluster centroid each iteration, or pixels might not develop a maximal membership degree to any cluster centroid.

A good result seemed to be achieved by applying SKM to the *pepper2* image, using the HSL with D_H color representation. At first glance this result seemed just as good as when using MEPSO, except that the amount of clusters discerned is on average 5.65, whereas MEPSO on average discerns 3.3 clusters —closer to the true amount of 3. The extra clusters have their members in lonely pixels at the “stem” of the “pepper”.

6.3.2 Cluster update rule

In Section 4.3.5 we noted that applying SKM on the *flower* image, using the HSL with D_{LSH} color representation, the typical and worst results seem to be in grayscale, but in

fact they are of a blueish color, all pixels having a very low saturation value. This did not change if we lowered the amount of clusters or change the value that an undefined hue component will have in the equation that computes the new cluster centroids.

It probably happens because most pixels in this image have a low saturation value, and the way a standard SKM algorithm updates its cluster centroids. Because the function that computes the new cluster centroids in Soft k-means uses a weighted average of the membership degree of a data point (to that cluster centroid) times the data point vectors themselves (Equation 2.6 on page 7):

$$\vec{C}_c \leftarrow \frac{\sum_{i=1}^n u_{ci}^m \cdot \vec{X}_i}{\sum_{i=1}^n u_{ci}^m}$$

the new cluster centroid values are likely to get a low saturation value. This results in an emphasis put on the lightness differences when calculating the new membership degrees, and because the image has many differences in lightness values, this suffices for “filling” all cluster centroids.

6.4 Parameters

6.4.1 Acceleration constants

For all three images, using the configuration of c_1 decreasing, c_2 increasing produced on average better resulting images than using the configuration of c_1 increasing, c_2 decreasing (the method used in the original MEPSO implementation). Not only does the configuration of c_1 decreasing, c_2 increasing produce better results than the configuration of c_1 increasing, c_2 decreasing, it also produces on average better results than when using no acceleration constants configuration at all ($c_1 = c_2 = 1$). We can explain this as follows.

If c_1 is zero and c_2 is above zero, the swarm acts as one stochastic hill-climber. If c_2 is zero and c_1 is larger than zero, the swarm effect is deleted and all particles act as independent hill-climbers. The case in which $c_1 < c_2$ is better for unimodal functions, that is, functions which have one global optimum. And the case in which $c_1 > c_2$ is better for multimodal functions, that is, functions having at least one global optimum [16].

The function to be optimized here is the Xie-Beni index (see Equation 2.10 on page 9), applied to the input image. Assuming the number of pixels in the image is larger than the number of possible cluster for the particles (which will probably be true in most real-life applications of clustering algorithms), we can presume this function to have one global optimum using our configuration settings. Thus, the problem is regarded as unimodal.

In the case of the original MEPSO algorithm, the swarm changes from being predominantly one stochastic hill-climber, to being predominantly a set of independent

hill-climbers. That is, the problem is first regarded as being predominantly unimodal, and later as being predominantly multimodal. This does not seem an obvious choice, given that the optimization problem is unimodal, so one should probably want to end the problem-solving with finding the global optimum. Also, by letting the particles movements be predominantly globally influenced in the beginning, the randomness of their initially assigned positions will play a larger role.

Our results have, indeed, shown that this is not a very good choice, and that the reverse configuration yields better results, even better than when using no acceleration constants at all. In the case of c_1 decreasing, c_2 increasing, the particles will move mainly independent in the beginning, so there is a higher chance of one particle finding a good position. During the second half of the search process, they will become more and more globally influenced, so that the best position found in the first half of the search process, will predominantly influence all particles, with eventually the particles converging.

6.4.2 Spatial window

Using MEPSO and the color representation HSL and D_H , clustering *pepper1* in the condition of using the spatial window, led to an increase in the space occupied by the pepper in the picture, that originally was white background (see Figure 4.39 on page 44). This phenomenon has already discussed in Section 6.2.1. The use of the spatial window merely encourages the underlying process.

When using the Soft K-means algorithm, the HSL with D_{LSH} color representation, and turning off the spatial window method, the algorithm produced notably worse results for *pepper2* compared to the situation in which the spatial window method was turned on. This might happen because SKM has less ability to overcome a bad start configuration. Here, it seems, the spatial window can keep pixels from wandering off to slightly more or equally luring cluster centroids.

Chapter 7

Conclusions and future research

7.1 Conclusions

Recall that our research question consisted of two parts. First and foremost we wanted to examine if the results of the so-called MEPSO algorithm of Das et al. [7] could be replicated for color images. The second part of our research question was if improvements could be made.

In this chapter we will first present a brief summary of our conclusions (Section 7.1.1). In Section 7.1.2, we will specifically look at the answer to the first part of our research question —if the results could be replicated. In Section 7.1.3 and Section 7.1.4 we will take a look at the improvements we introduced. We have looked at improvements in both the search space (i.e. the color representation used) and the configuration of the acceleration constants (which affects the interdependence over time of the particles in the swarm). Finally, in Section 7.1.5 we will also consider how this PSO algorithm compares to Soft k-means clustering for this task.

7.1.1 Summary

If we look at all three factors —the algorithm (MEPSO or SKM), the color representation (RGB, HSL with D_{LSH} , HSL with D_{H}), and the parameters (acceleration constants, spatial window method) —what would be the best combination of choices to use for color image segmentation?

If one would want to segment multiple, different images, without time being a crucial factor, the best results in general would be achieved by using MEPSO, with the HSL with D_{H} color representation, and the acceleration constants configuration of “ c_1 decreasing, c_2 increasing”, not using the spatial window method.

If one already knows the number of clusters the image should have, and the result needs to be produced as fast as possible, using SKM with the HSL with D_{H} color representation, without the spatial window method, would probably be the best choice.

If the image contains many pixels in pure grayscale, the distance measure D_{H} is better not used. However, for images in which the majority of pixels is in grayscale, using a

color image clustering algorithm does seem inappropriate anyway —the best for those images would probably be to convert them to a full grayscale image, and use one of the abundantly available algorithms for clustering grayscale images. However, lots of images in which the majority of pixels are colored, will contain some pixels in pure grayscale, for example because of very strong highlights or very dark shadows. One way that one could deal with images that contain a certain amount of grayscale pixels, is to include a preprocessing step whereby, if a pixel has no saturation, it gets a very low, but non-zero saturation value, and a random or averaged hue value.

7.1.2 Comparison with original MEPSO

We can conclude that the results of Das et al. [7] could be replicated for color images. We used a pepper image and texture image highly similar to their pepper image and texture image, the same RGB color representation, and an algorithm as similar as possible. In Section 4.1 can be seen that we achieved comparable or better visual results in the best cases. In Section 5.1 can be seen that our MEPSO implementation also yielded better or comparable results on the validity indices.

7.1.3 Color representations

We tested three color representations: RGB, HSL with D_{LSH} , and HSL with D_H .

The RGB color space performs worst in all cases. It performs worst on two of the three validity indices in the data results for the color representations in Section 5.2, and the visual results it yields in Section 4.2 are also the worst of all three color representations used.

From the data results for the color representations in Section 5.2 it can not be inferred if either the color representation HSL with D_H or the color representation HSL with D_{LSH} performs superior. However, in Section 4.2 it can be seen that the HSL with D_H color representation produces the best visual results on average, but especially for images that contain lighting effects.

However, using the HSL representation with our D_H distance measure, there is a drawback when the image includes pixels in pure grayscale (i.e. pixels with no saturation, and therefore, an undefined hue component). Using the D_{LSH} distance measure we can still use the lightness component if the hue component is undefined. But when using the D_H distance measure, where the distance between an undefined hue component and any other hue component is regarded as zero, the algorithm will gradually assign these pixels to other clusters, therefore the non-hue area in the image will become gradually “occupied”, or “occupy” other areas in the image.

The *flower* image is held up as an example of the superiority of the D_{LSH} distance measure by Patrascu [15], but he used this distance measure in a SKM algorithm with a gradient descent method (based on this distance measure) for updating the cluster centroids. Our results show that a good clustering solution using the D_{LSH} distance measure could not be replicated using a PSO algorithm with the Xie-Beni index as cluster validity index.

7.1.4 Parameters

Acceleration constants

In Section 4.4.1 we saw that the setting of c_1 decreasing, c_2 increasing yielded better or comparable resulting images in the “Typical result” and “Worst result”-cases, compared to the setting without using acceleration constants ($c_1 = c_2 = 1$), as well as to the setting of c_2 decreasing, c_1 increasing (as used in the original MEPSO implementation). For the “Best result”-case it yielded also superior or comparable images, except when the image was the *flower* image. The best “Best result”-case for the *flower* image was achieved using the setting c_2 decreasing, c_1 increasing.

In Section 5.2 we can see that the c_2 decreasing, c_1 increasing setting indeed performed worse on all three validity indices. The setting $c_1 = c_2 = 1$ had better results than the setting of c_1 decreasing, c_2 increasing for the indices of the Xie-Beni index and the Partition Entropy. However, these differences were relatively much smaller than the difference between the Cluster Difference Error of both settings (on which c_1 decreasing, c_2 increasing was better).

We can recommend the c_1 decreasing, c_2 decreasing configuration for getting better average results in all cases. We used the color representation HSL with D_H to test the acceleration constant settings, but we can safely generalize this to other color representations, since the color representation is irrelevant to these settings.

Spatial window

In Section 3.3.1 (page 23) we explained that a good investigation of the spatial window method compared to how it performed in the original MEPSO was not possible because we do not know the relative size of the window compared with the input images used in the original experiment.

We did not examine how it affected the validity indices in Chapter 5. We did look at the visual results turning off the spatial window produced for the images *pepper1* and *pepper2*. From this we can infer that it is better not to be used when the color representation is HSL with D_H and the image will contain relatively many pixels in grayscale (i.e. pixels with an undefined hue component).

Because in all other cases (using MEPSO and one of the two other color representations), for *pepper2*, turning the spatial window method on did not produce distinguishably different resulting images, we can, at the least, question if a *windowSize:imageSize* ratio of 1:400 (a 5×5 pixel window used on an image of 100×100 pixels) provides any benefit within MEPSO.

7.1.5 Comparison with Soft K-means

In Section 5.3 can be seen that our MEPSO implementation outperforms SKM on all three validity indices, but the Partition Entropy result does not differ much between SKM and MEPSO.

As discussed in Section 6.3.1, Soft k-means profoundly suffers from the fact it has to use a fixed amount of cluster centroids, which in our cases was too large. Therefore, it failed to produce a clustered image in most cases for *simpleBdiff* and *simpleSBdiff* (see Section 4.3), regardless of the color representation used. For the other images, the visual results obtained using SKM were too detailed.

If one averages over the data vectors and their membership degrees to compute the new cluster centroids, as in Equation 2.6:

$$\vec{C}_c \leftarrow \frac{\sum_{i=1}^n u_{ci}^m \cdot \vec{X}_i}{\sum_{i=1}^n u_{ci}^m}$$

some odd results may occur, as discussed in Section 6.3.2.

In in Section 4.3 can also be seen that when using SKM, the results for each image were much more consistent compared to the results when using MEPSO. Consistency is not necessarily an advantage, though. Especially in problems with many local optima, like when clustering images, this could mean that SKM gets stuck in the same local optimum every time.

7.2 Future research

7.2.1 Spatial window

The effect of the spatial window method is currently not clear. This could be investigated. Doing this, one could especially investigate the optimal size of this window, given the size of the image—the authors of the original MEPSO did mention they used a window of 5×5 pixels, but did not mention the dimensions of the images they used.

7.2.2 Run-time complexity

For a better performance measure of MEPSO when it comes to run-time, one could look at how long it takes for MEPSO to achieve a respectable result, instead of using a fixed number of iterations. To accomplish this, one could use a stopping criterion based on a minimum threshold of change in the cluster centroids of the particles. In this case, one should also investigate the optimal population size of MEPSO, since this population size is a large time-consuming factor.

A time-consuming factor of both MEPSO and SKM is the fact it needs to calculate the membership matrix \mathbf{U} with membership degrees for every (active) cluster centroid of every particle each iteration. If the spatial window method is used, it also needs to recalculate this matrix every cluster centroid of every particle each iteration. However, if the spatial window method is not used, there is no requirement in MEPSO that the membership degrees of the data points to the cluster centroids should be fuzzy. An adaptation of MEPSO, without a spatial window method and based on k-means instead

of Soft k-means, would consume less memory resources and also need less computations than the original MEPSO. It would be interesting to investigate if it yields results just as well as this MEPSO implementation.

7.2.3 Network topology

Here, we used a swarm with a global best neighborhood topology. The strength of a swarm algorithm however, lies partially in the fact that the particles have the possibility of only incorporating information of a selected subset of particles. That means that the global best position they use in the update rule to determine their new position, is based on the best position found so far of only a selected subset of the swarm. Using only a subset of the total swarm in computing the movement for each particle, would result in subsets of the swarm following their common “local” global best, thus it would likely make the total swarm diverge more in the search space.

A neighborhood topology can be geographically based or communication based. In a communication based topology, the neighbors of a particle are pre-determined. Geographically based neighborhood topologies are based on the Euclidean proximity of the particles to each other in the search space. Of course, the last topology is more computationally expensive.

7.2.4 Fitness function

The fitness function used in this algorithm was based on the Xie-Beni index (see Section 2.2.3). As we noted in Chapter 5, this index does not really provide a proper measure of the quality of the clustering in the resulting image. There are a variety of other cluster validity indices available, and it could be investigated which one would work best for PSO-based clustering of color images. Devising a new validity index specifically tailored to color image segmentation is also possible, and note that one has a large freedom in this respect because the validity index does not have to be differentiable when used as a fitness function in PSO. Another possibility is to adapt the Xie-Beni index. For example, a peculiar attribute of the Xie-Beni index is that it consistently tends to favor clustering solutions having a fewer number of (active) clusters. We hereby suggest a possible improvement by adding the amount of clusters as a factor in the denominator, the new index being:

$$XB_{new} = \frac{\sum_{c=1}^k \sum_{i=1}^n u_{ci}^2 \cdot \|\vec{X}_i - \vec{C}_c\|^2}{k \cdot n \cdot \min_{c \neq d} \|\vec{C}_c - \vec{C}_d\|^2} \quad (7.1)$$

7.2.5 Color representation

RGB, HSV and HSL are not the only color models available. CIE is a color system developed for representing psychophysical uniformity [5]. From it, the CIE(Lab) and

CIE(Luv) color spaces can be constructed. CIE(Luv) has been known to yield good results when used for color image segmentation [19].

7.3 Concluding remarks

Particle Swarm Optimization (PSO) is a relatively young optimization technique currently being used in many problem domains. PSO being a metaheuristic with the choice of the parameters having a large effect on its performance makes it hard to evaluate as a general optimization technique [16] (but see [20]). In this thesis we have investigated the application of PSO to the specific problem domain of color image segmentation.

We have examined different PSO configuration settings in relation to the task of color image segmentation. We have concluded that a PSO-based algorithm can serve as a suitable method for color image segmentation. We have demonstrated some configurations that can be made to the general PSO algorithm to optimize it for this task, and we have outlined under what circumstances such a PSO-based algorithm should be preferred over a more traditional Soft k-means-based algorithm (see Section 7.1.1).

As we have outlined in Section 7.2 there are still many possibilities for future research of PSO-based color image segmentation algorithms, pertaining to properties of both the general PSO algorithm and color theory. A more theoretical analysis of performance and convergence properties when using a PSO-based color image segmentation algorithm could also be conducted.

References

- [1] Ajith Abraham, Swagatam Das, and Sandip Roy. Swarm intelligence algorithms for data clustering. In Oded Maimon and Lior Rokach, editors, *Soft Computing for Knowledge Discovery and Data Mining*, pages 279–313. Springer US, 2008. ISBN 978-0-387-69935-6.
- [2] M.K. Agoston. *Implementation and algorithms*, pages 299–307. Computer graphics and geometric modeling. Springer, 2005. ISBN 9781852338183.
- [3] James Blondin. Particle swarm optimization: A tutorial. http://cs.armstrong.edu/saad/csci8100/pso_tutorial.pdf, sep 2009.
- [4] Urszula Boryczka. Ant clustering algorithm. *Intelligent Information Systems*, pages 377–386, 2008.
- [5] H. D. Cheng, X. H. Jiang, Y. Sun, and Jing Li Wang. Color image segmentation: Advances and prospects. *Pattern Recognition*, 34:2259–2281, 2001.
- [6] Tai Wai Cheng, Dmitry B. Goldgof, and Lawrence O. Hall. Fast fuzzy clustering. *Fuzzy Sets and Systems*, 93(1):49–56, 1998. ISSN 0165-0114.
- [7] Swagatam Das, Ajith Abraham, and Amit Konar. Spatial information based image segmentation using a modified particle swarm optimization algorithm. *International Conference on Intelligent Systems Design and Applications*, 2:438–444, 2006.
- [8] Marco Dorigo, Mauro Birattari, and T. Stützle. Ant Colony Optimization: Artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.
- [9] S. A. Elavarasi, J. Akilandeswari, and B. Sathiyabhama. A survey on partition clustering algorithms. *International Journal of Enterprise Computing and Business Systems*, 1(1):1–14, 2011.
- [10] Greg Hamerly and Charles Elkan. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the eleventh international conference on Information and knowledge management, CIKM '02*, pages 600–607. ACM, 2002.
- [11] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Survey*, 31(3):264–323, 1999. ISSN 0360-0300.

- [12] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, WA, Australia, 1995. IEEE Press. ISBN 0-7803-2768-3.
- [13] Erik D. Lumer and Baldo Faieta. Diversity and adaptation in populations of clustering ants. In *Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3: from animals to animats 3*, SAB94, pages 501–508, Cambridge, MA, USA, 1994. MIT Press. ISBN 0-262-53122-4.
- [14] V.K. Panchal, Harish Kundra, and Jagdeep Kaur. Comparative study of particle swarm optimization based unsupervised clustering techniques. *International Journal of Computer Science and Network Security*, 9(10):132–140, 2009.
- [15] Vasile Patrascu. New fuzzy color clustering algorithm based on hsl similarity. In João Paulo Carvalho, Didier Dubois, Uzay Kaymak, and João Miguel da Costa Sousa, editors, *Proceedings of the Joint 2009 International Fuzzy Systems Association World Congress and 2009 European Society of Fuzzy Logic and Technology Conference*, pages 48–52, Lisbon, Portugal, 2009.
- [16] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1:33–57, 2007. ISSN 1935-3812.
- [17] A. Salem, B. Samma, and R. Abdul. Adaptation of k-means algorithm for image segmentation. In *Proceedings of World Academy of Science, Engineering and Technology*, volume 38, pages 58–62, 2009.
- [18] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proceedings of Congress on Evolutionary Computation*, pages 79–73, 1998.
- [19] W. Skarbek and A. Koschan. *Colour image segmentation: a survey*. Technical report, Institute for Technical Informatics. Technical University of Berlin, 1994.
- [20] Frans Van Den Bergh. *An analysis of particle swarm optimizers*. PhD thesis, Pretoria, South Africa, 2002.
- [21] Christian Veenhuis and Mario Köppen. Data swarm clustering. In *Swarm Intelligence in Data Mining*, volume 34/2006 of *Studies in Computational Intelligence*, pages 221–241. Springer Berlin / Heidelberg, 2006.
- [22] Xuanli Lisa Xie and Gerardo Beni. A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):841–847, 1991. ISSN 0162-8828.

Appendices

Appendix A

Code

A.1 Conversion between HSL and RGB

Listing A.1: RGB to HSL conversion, Matlab code

```
1 function HSLpic=rgb2hsl( RGBpic)
2 %input: rgb image matrix, M*N*3 in [0,1][0,1][0,1]
3 %output: hsl image matrix, M*N*3 in [0,360][0,1][0,1]
4 %
5 % (C) Agnes van Belle , 2011
6 % using
7 % - an implementation by Vladimir Bychkovsky, June 2008
8 % - pages 305-306 of Implementation and Algorithms, Agoston, M.K.,2005 ([2])
9
10 Xmatrix=reshape( RGBpic, [], 3) ;
11
12 R = Xmatrix(:,1) ;
13 G = Xmatrix(:,2) ;
14 B = Xmatrix(:,3) ;
15
16 Mx = max(Xmatrix, [], 2) ;
17 Mn = min(Xmatrix, [], 2) ;
18
19 %compute lightness
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 L = (Mx+Mn)/2;
22
23 %compute saturation
24 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
25 S = zeros( size(L) );
26
27 nohue_idx = (Mx == Mn); % if no saturation, hue is undefined
28 S(nohue_idx) = 0;
29
30 Dist = (Mx-Mn);
31
32 Val = Dist./(Mx+Mn);
33 idx = ~nohue_idx & L <= 0.5;
34 S(idx) = Val(idx);
35
36 Val = Dist./(2-Mx-Mn);
37 idx = ~nohue_idx & L > 0.5;
38 S(idx) = Val(idx);
```

```

39
40 %compute hue
41 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
42 rismax_idx = R == Mx;
43 gismax_idx = G == Mx;
44 bismax_idx = B == Mx;
45
46 H = zeros(size(L));
47
48 Val = (G-B)./Dist;
49 H(rismax_idx) = Val(rismax_idx);
50 Val = 2 + ((B-R)./Dist);
51 H(gismax_idx) = Val(gismax_idx);
52 Val = 4 + ((R-G)./Dist);
53 H(bismax_idx) = Val(bismax_idx);
54
55 H = H .*60;
56 Hneg_idx = H < 0;
57 H(Hneg_idx) = H(Hneg_idx) + 360;
58
59 H(nohue_idx) = NaN; % NaN = UNDEFINED
60
61 HSLX = [H,S,L];
62 HSLpic=reshape(HSLX, size(RGBpic));

```

Listing A.2: HSL to RGB conversion, Matlab code

```

1 function RGBpic=hsl2rgb(HSLpic)
2 %input:hsl image matrix, M*N*3 in [0,360][0,1][0,1]
3 %output:rgb image matrix, M*N*3 in [0,1][0,1][0,1]
4 %
5 % (C) Agnes van Belle, 2011
6 % using
7 % - an implementation by Vladimir Bychkovsky, June 2008
8 % - pages 305-306 of Implementation and Algorithms, Agoston, M.K.,2005 ([2])
9
10 Xmatrix=reshape(HSLpic,[],3);
11
12 [length,width]=size(Xmatrix);
13 RGBX=zeros(length,width);
14
15 H = Xmatrix(:,1);
16 S = Xmatrix(:,2);
17 L = Xmatrix(:,3);
18
19 % set H in [0,6)
20 hismax_idx = (H==360);
21 H(hismax_idx) = 0;
22 H = H./60;
23
24 % make R,G & B values same as lightness-value, if no saturation
25 siszero_idx = (S == 0);
26 hisundef_idx = isnan(H); % should be the same, but check for sure
27 if(not(sum(siszero_idx - hisundef_idx) == 0))
28     sprintf('invalid unput for hsl2rgb');
29 end
30
31 nohue_idx = hisundef_idx & siszero_idx;
32 Val = L(nohue_idx);
33 XRGB(nohue_idx,1) = Val;
34 XRGB(nohue_idx,2) = Val;
35 XRGB(nohue_idx,3) = Val;
36
37 lowl_idx = L <= 0.5;
38 V = (L.*(1+S)).*lowl_idx + ((L+S) - L.*S).*(~lowl_idx);
39
40 Min = 2.*L-V;
41 SV = (V-Min)./V;
42 Sextant = floor(H);
43 Fract = H-Sextant;
44 Vsf = V.*SV.*Fract;
45 Mid1 = Min+Vsf;
46 Mid2 = V-Vsf;
47
48 for i=1:length
49     if(not(V(i) == 0))
50         C = zeros(1,3);
51         sextant = Sextant(i);
52         switch sextant
53             case 0
54                 C = [V(i),Mid1(i),Min(i)];
55             case 1
56                 C = [Mid2(i), V(i), Min(i)];
57             case 2
58                 C = [Min(i), V(i), Mid1(i)];
59             case 3
60                 C = [Min(i), Mid2(i), V(i)];

```

```
61     case 4
62         C = [Mid1(i), Min(i), V(i)];
63     case 5
64         C = [V(i), Min(i), Mid2(i)];
65     end
66     RGBX(i,:) = C;
67 end
68 end
69
70 RGBpic = reshape(RGBX, size(HSLpic));
```

A.2 HSL distance measures

Listing A.3: Distance measure D_H , Matlab code

```
1 function d = distance(C1,C2)
2 %input: two 1x3 vectors in [0,1][0,1][0,1]
3 %output: a scalar
4
5     h1 = (C1(1).*360);
6     h2 = (C2(1).*360);
7
8     diff_abs = abs(h1-h2);
9     hdist = min(diff_abs , 360-diff_abs) / 180;
10
11 % (0 * NaN) == NaN, in Matlab
12 if (isnan(h1) || isnan(h2)) % one or both UNDEFINED
13     d = 0;
14 else
15     d = hdist;
16 end
17
18 d = d^2;
19 end
```

Listing A.4: Distance measure D_{LSH} , Matlab code

```

1 function d = distance(C1,C2)
2 %input: two 1x3 vectors in [0,1][0,1][0,1]
3 %output: a scalar
4
5 h1 = (C1(1).*360);
6 h2 = (C2(1).*360);
7
8 s1 = C1(2);
9 s2 = C2(2);
10
11 l1 = C1(3);
12 l2 = C2(3);
13
14 diff_abs = abs(h1-h2);
15 hdist = min(diff_abs , 360-diff_abs) / 180;
16
17 s1 = s1 * (pi/2);
18 s2 = s2 * (pi/2);
19
20 ldist = abs(l1-l2);
21
22 c1 = sin(s1); % chromaticity
23 c2 = sin(s2);
24 a1 = cos(s1); % achromaticity
25 a2 = cos(s2);
26
27
28 % (0 * NaN) == NaN, in Matlab
29 if (isnan(h1) || isnan(h2)) % one or both are NaN (= UNDEFINED)
30     d = ldist;
31 else
32     d = a1*a2*ldist + c1*c2*hdist;
33 end
34
35 d = d^2;
36 end

```