Nonlinear Clustering on Sparse Grids

Interdisciplinary Project (IDP)

Julius Adorf, Technische Universität München

August 13, 2012

Abstract

This work applies a recent sparse-grid-based spectral clustering method to the problem of unsupervised image segmentation. The applicability of the method is tested on synthetic images, as well as on images from the Berkeley Segmentation Dataset. The impact of parameters is examined, focusing on the level of the sparse grid, the number of clusters, and the width of the similarity kernel. Serving as an automated alternative to manual model selection, the balanced linefit criterion is tested. The effects of hierarchical surplus refinement are studied and compared with variants introduced in this work, aiming at achieving good results with few grid points.

Contents

1 Introduction

1.1 Problem

Clustering methods have been applied to many problems in medicine, biology, business, and computer science. As an unsupervised machine learning technique, cluster analysis discovers structure in data without having prior knowledge gathered on a ground truth dataset.

A particular application of clustering is the segmentation of images, which is a common, but difficult task in image analysis and computer vision. In image segmentation, cluster analysis shall find groups of pixels. What kind of grouping is desirable varies widely with the intended purpose. This technical report focuses on images from the Berkeley Segmentation Dataset, such as shown in Figure 1 (left). A desirable segmentation is presented in Figure 1 (right), where the boundaries closely match the boundaries perceived by humans.

Images contain much data. The original picture of Figure 1 (left) consists of $481 \times 321 \approx 150.000$ pixels. A nonlinear, spectral clustering approach to image segmentation is presented by Shi and Malik [1], who cluster the pixels by solving an eigensystem. The size of the eigensystem is given by the number of pixels.

The data in a digital image is usually arranged in a full two-dimensional grid. For 3D medical imaging, the full grid even spans three dimensions. The number of grid points in full grids of higher dimensions render the spectral decomposition of a data-derived

Figure 1: A test image from the Berkeley Segmentation Dataset (left), and a possible manual segmentation (right).

similarity matrix infeasible.

Peherstorfer et al. [2] propose a spectral clustering approach where eigenfunctions on a sparse grid are learned and can be evaluated at any other point in the domain. The key is that the size of the eigensystem depends on the number of sparse grid points rather than the number of data points. This is a prospective method to counter the so-called curse of dimensionality.

In this technical report, the image segmentation problem serves as an application of the spectral clustering method on sparse grids, as proposed by Peherstdorfer et al. [2]. Their proposed method and their corresponding implementation is applied, evaluated, and extended in this work.

Section 2 summarizes the required theory that leads to the spectral clustering method by Peherstorfer et al. [2]. Section 3 describes the actual method, and the extensions added in this work. Section 4 presents the results, and draws conclusions.

1.2 Related Work

In some datasets, such as the synthetic one shown in Figure 2, the clusters cannot be separated by hyperplanes. In these cases, more powerful, nonlinear methods are sought after. The spectral clustering algorithm by Shi and Malik [1] approximates an optimal normalized cut in a similarity graph derived from the data. Belkin and Niyogi [3] introduce the closely related Laplacian Eigenmaps for dimensionality reduction, where data is non-linearly mapped to a low-dimensional latent space. In that space, linear clustering methods can be applied again. Alzate and Suykens [4] formulate spectral clustering in terms of weighted kernel principal components analysis, and keep computational costs low by only solving the involved eigensystem on a subsample of the data. The remaining out-of-sample data is mapped into latent space by projecting it onto the eigenvectors. Another out-of-sample extension is presented by Fowlkes et al. [5], who assign out-of-sample data to clusters via eigenfunctions based on the subsample.

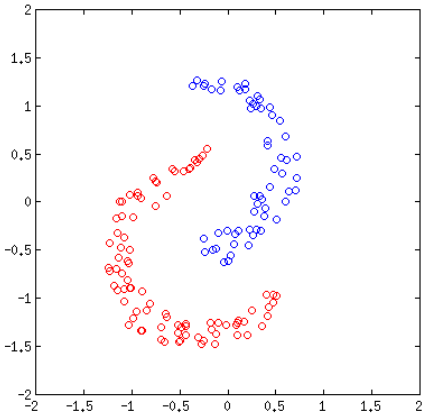

Figure 2: A synthetic, not linearly separable, dataset showing two moons, clustered with the spectral method as described by Alzate and Suykens [4].

However, if the data, as in images or volumetric datasets, comes from a full grid, the size of the eigensystems in the methods by Alzate and Suykens [4], as well as by Fowlkes et al. [5] grows linearly with the number of data points in the subsample. In contrast, this work builds upon the spectral clustering approach by Peherstorfer et al. [2], who solve an eigensystem for the coefficients of sparse-grid-based eigenfunctions. As mentioned before, the size of the involved eigensystem here depends on the number of points in the sparse grid rather than the size of the subsample.

The spectral clustering methods mentioned in this report are all based on some modified similarity matrix. For image segmentation, building a similar-

ity matrix requires a useful similarity measure between pixels. Fowlkes et al. [5], as well as Alzate and Suykens [4] use local color histograms computed on color-quantized images. This path is also taken in this work. For background on color quantization, see the work by Heckbert [6], and by Wu [7].

Martin et al. [8] provide the Berkeley Segmentation Dataset 300 (BSDS300), which contains a set of training images, and a set of test images. The dataset is equipped with ground truth data, which has been obtained from manual segmentations.

2 Prerequisites

The sparse-grid-based spectral clustering approach using Laplacian Eigenmaps [2] builds upon many existing concepts. This section summarizes the theory behind these concepts. It comprises cluster analysis (Section 2.1), image segmentation (Section 2.2), spectral clustering (Section 2.3), and sparse grids (Section 2.4).

2.1 Clustering

In clustering, a dataset $X = \{\mathbf{x}^t\}_{t=1}^M$ shall be partitioned into k clusters, such that similar points end up in the same cluster, and dissimilar points are separated into different clusters.

First, it is curious that the number of clusters is part of the question rather than part of the answer. Indeed, some clustering applications determine the ideal number of clusters through automatic model selection, whereas in other applications, the number of clusters needs to be manually specified.

Second, the objective function needs to be formulated. To this end, let us define a a similarity measure $s_{ij} \geq 0$ that specifies the similarity between the i-th and the j-th data point. An objective function then maximizes intra-cluster similarity and minimizes inter-cluster similarity. The exact method, and the pursued trade-off between these two goals is application-dependent.

2.2 Image Segmentation

This work focuses on image segmentation through clustering. In such a case, the dataset is formed by the pixels. The choice of similarity between the pixels is non-trivial.

One possible choice is to compute the $\mathcal{X}^2$ distance between local color histograms, and then to weight the distance with a radial basis function kernel in order to obtain a similarity measure. This is the path taken by Fowlkes et al. [5], which we will follow in this work.

First, the image is quantized into B colors such that each color histogram has B bins. The histograms are computed within a $W \times W$ square neighborhood. The value h_{ib} denotes the b-th bin value for the i-th pixel. When computing these histograms, a small positive value is added to empty bins such that $h_{ib} > 0$. Then, distances w_{ij} between pixels can be obtained by Equation 1.

$$w_{ij} = \frac{1}{2} \sum_{b=1}^{B} \frac{(h_{ib} - h_{jb})^2}{h_{ib} + h_{jb}} \tag{1}$$

Second, the weights w_{ij} are fed into a radial-basis function with parameter σ.

$$s_{ij} = \exp\left(-\frac{w_{ij}}{\sigma}\right) \tag{2}$$

2.3 Spectral Clustering

Spectral clustering methods are based on the solution of an eigensystem. They aim at partitioning the *similarity graph*, where the data points form the nodes, and the similarity measure assigns weights to the edges.

Spectral clustering can be derived from the normalized cut criterion [1]. Unlike the global minimum cut, the normalized cut penalizes unbalanced clusters, instead of only minimizing the sum of edge weights in the cut. Finding the optimal normalized cut in the similarity graph is NP-complete [1].

One particular spectral clustering approach is presented here, because it lays ground for Section 3.1. Assume the similarity graph to be connected. Let $\mathbf{S} \in \mathbb{R}^{M \times M}$ the adjacency matrix of the similarity

graph, called the *similarity matrix*. Let the diagonal matrix $\mathbf{D} \in \mathbb{R}^{M \times M}$ be the *degree matrix* formed by the row sums of $\mathbf{S}$. Then, playing a central role, the *graph Laplacian* $\mathbf{L} \in \mathbb{R}^{M \times M}$ is defined by $\mathbf{L} = \mathbf{D} - \mathbf{S}$.

In the normalized spectral clustering algorithm [9], an approximative solution to the normalized cut is obtained by minimizing the generalized rayleigh quotient (see [3, 9])

$$\min_{\mathbf{v}} \frac{\mathbf{v}^T \mathbf{L} \mathbf{v}}{\mathbf{v}^T \mathbf{D} \mathbf{v}} \quad \text{s.t.} \quad \mathbf{v}^T \mathbf{D} \mathbf{1} = 0 \wedge ||\mathbf{v}||_2 = d \tag{3}$$

where d is some constant, and $\mathbf{1}$ denotes the vector of ones. This minimization problem is solved by computing the eigenvector $\mathbf{v}^{(2)}$ corresponding to the second-smallest eigenvalue of the generalized eigensystem (see [9])

$$\mathbf{L}\mathbf{v} = \lambda \mathbf{D}\mathbf{v}. \tag{4}$$

The final cluster assignment is obtained by thresholding the eigenvector $\mathbf{v}^{(2)}$ at zero. For example, the i-th data point is assigned to the first cluster if and only if the component $\mathbf{v}_i^{(2)}$ is negative.

The scheme can be extended to multiway spectral clustering. Let the eigenvectors $\mathbf{v}^{(j)}, j \in 1...M$ of Equation 4 be sorted by eigenvalue in ascending order. The first eigenvalue is zero, the corresponding eigenvector $\mathbf{v}^{(1)}$ is the vector of ones, and hence is of no interest. Let $\mathbf{A} = \left[\mathbf{v}^{(2)} ... \mathbf{v}^{(k)}\right]$ be the $\mathbb{R}^{M \times (k-1)}$ matrix obtained by stacking the eigenvectors corresponding to the $k-1$ smallest non-zero eigenvalues horizontally. The entries of matrix $\mathbf{A}$ are called *score variables*, and they provide a new low-dimensional description of the data.

In the last step, *reclustering* the rows of matrix $\mathbf{A}$ provides the final clustering assignment, for example by using k-means. This is possible because similar points in the original space are mapped to nearby points in low-dimensional space [3].

2.4 Sparse Grids

This section gives a brief introduction into the principles of sparse grids. For a comprehensive treatment of sparse grids, see the work by Bungartz and Griebel [10]. A shorter introduction is given by Gerstner and Griebel [11].

Treating a function numerically on a digital computer requires a discretization scheme. For example, a digital camera discretizes an image by sampling the image in each dimension in regular intervals, forming a full grid. The intensity values are stored at the points of the full grid.

A full grid is subject to the curse of dimensionality. If some function in d dimensions is modeled on a full grid with N grid points in each dimension, then the total number of grid points is N^d.

However, a function on a sparse grid can approximate a function on a full grid. Note that in this work, we don't want to find another way to represent the image, but rather find eigenfunctions that help finding good clusters. Using sparse grids, the number of grid points can be significantly reduced. In the following, only 2D sparse grids will be discussed.

Sparse grids are constructed hierarchically. Regular sparse grids exist at different levels, and each level adds new grid points in a principled manner. For example, a sparse grid of level 1 consists of only one grid point in the center of the domain, as shown in Figure 3 (a). A sparse grid of level 2 consists of 5 grid points: the level 1 grid, plus the grid points shown in Figure 3 (b, c). A level 3 grid is depicted in Figure 5, and the triangular scheme can be extended to grids of higher levels.

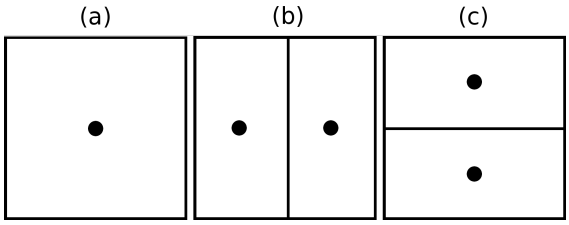

Figure 3: The grid points of a regular sparse grid of level 2, and the supports of the corresponding basis functions.

A basis function is centered at each grid point. The support of these basis functions varies with the level at which the grid points were introduced. In the classic approach, the basis functions are the products of

simple hat functions. For example, the grid points shown in Figure 3 have corresponding pagoda-shaped basis functions as depicted in Figure 4.

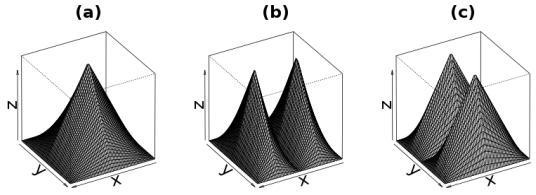

Figure 4: The pagoda-shaped basis functions of a regular sparse grid of level 2.

The sparse grids that have been discussed so far assume a homogeneous zero boundary. There are different ways to treat the boundary in cases, where this assumption does not hold. In this work, we use a linear boundary grid.

Sparse grids can be adaptively refined in regions that require higher resolution. This will be discussed in Section 3.2.

We use `SG++`[1] as a framework for sparse grids. It offers regular grids, adaptively refined grids, various boundary types, and efficient routines for evaluating the grid functions.

3 Method

3.1 Spectral Clustering on Sparse Grids

Peherstorfer et al. [2] introduce a new spectral clustering approach on sparse grids. The score variables are generated by eigenfunctions. These eigenfunctions are based on the sparse grid.

It is sadly infeasible to apply the classic spectral clustering approach discussed in Section 2.3 straightaway to all pixels in the image. Peherstorfer et al. [2] address this problem in two steps: First, the data is split into a small training set and a large testing set such that the expensive operations are computed on the training set only, and the testing set is treated cheaply by an out-of-sample extension. Second, instead of the score variables (see Section 2.3), it is the coefficients of sparse-grid-based eigenfunctions that are computed from an eigensystem. The obtained eigenfunctions can then be evaluated efficiently in order to generate the score variables for both the training set and the testing set.

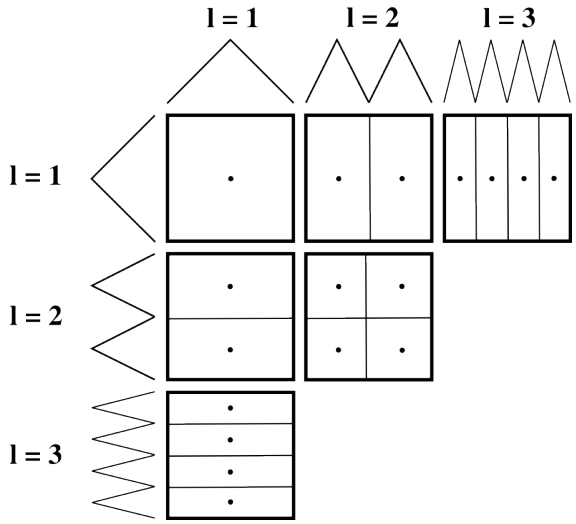

Figure 5: A triangular scheme that shows the construction of a sparse grid in two dimensions. The simple hat functions are used to construct basis functions for each grid point. The supports of these basis functions are shown in the figure.

On two-dimensional domains and with two-way clustering, the intuition behind this approach can be illustrated by looking at one eigenfunction. Such an eigenfunction can be visualized as a mountain landscape. This eigenfunction should have similar function values at locations where pixels are similar. In this report, the k-means algorithm is used for reclustering the score variables. As such, in this example with only one eigenfunction, k-means would find isolines in this mountain landscape that separate the two clusters. The question remains on how to obtain eigenfunctions with such qualities.

Here is the method from Peherstorfer et al. [2], described in the context of the two-dimensional domain

[1] http://www5.in.tum.de/SGpp

of images. Note that this method includes a regularization term, which we omit in this discussion. The method works as follows: let $M \in \mathbb{N}$ be the number of pixels in the training set. Let $N \in \mathbb{N}$ be the number of grid points in the sparse grid, which is also the number of basis functions $\phi_n : \mathbb{R}^2 \to \mathbb{R},\ n \in 1 \dots N$. The vector $\boldsymbol{\alpha} \in \mathbb{R}^N$ denotes the sought-after coefficients for these basis functions. The wanted function on the sparse grid then can be written as

$$\mathrm{f}(\mathbf{x}) = \sum_{n=1}^{N} \boldsymbol{\alpha}_n \Phi_n(\mathbf{x}) \tag{5}$$

Since the function f is linear in the coefficients, Equation 5 can be rewritten for the i-th training location $\mathbf{x}_i$ as $\mathrm{f}(\mathbf{x}_i) = \mathbf{b}_i^T \boldsymbol{\alpha}$, where $\mathbf{b}_i^T = (\Phi_1(\mathbf{x}_i) \dots \Phi_N(\mathbf{x}_i))$. Hence, if the vectors $\mathbf{b}_i^T$ form the rows of a matrix $\mathbf{B} \in \mathbb{R}^{M \times N}$, then the function values $\mathbf{v} \in \mathbb{R}^M$ of at the training locations can be written as

$$\mathbf{v} = \mathbf{B}\boldsymbol{\alpha}. \tag{6}$$

Plugging this into Equation 3 gives

$$\min_{\boldsymbol{\alpha}} \frac{\boldsymbol{\alpha}^T \mathbf{B}^T \mathbf{L} \mathbf{B} \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T \mathbf{B}^T \mathbf{D} \mathbf{B} \boldsymbol{\alpha}} \tag{7}$$

This way, one arrives at the generalized eigensystem proposed by Peherstorfer et al. [2, Equation 6]

$$\mathbf{B}^T \mathbf{L} \mathbf{B} \boldsymbol{\alpha} = \lambda \mathbf{B}^T \mathbf{D} \mathbf{B} \boldsymbol{\alpha}. \tag{8}$$

In this report, the eigensystem in Equation 8 is solved for the $k-1$ eigenvectors that correspond to the smallest positive eigenvalues. These eigenfunctions are then evaluated for both training data and testing data, giving the score variables as the input to the k-means algorithm.

The straight-forward implementation of Equation 8, as it is used in this report, constructs the system matrices $\mathbf{B}^T\mathbf{L}\mathbf{B}$ and $\mathbf{B}^T\mathbf{D}\mathbf{B}$ explicitly. Figure 6 sketches the matrix $\mathbf{B}^T\mathbf{L}\mathbf{B}$ in order to emphasize the involved matrix dimensions. Both matrices are fed into a generalized eigensystem solver implemented in the GNU Scientific Library under the name `gsl_eigen_genv`. The matrix product $\mathbf{B}^T\mathbf{L}\mathbf{B}$ is in $O(M^2N + MN^2)$ time, whereas the product

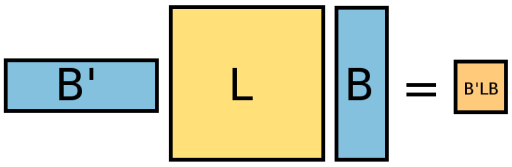

Figure 6: A schematic figure showing the sizes of the matrices $\mathbf{B}^T$, $\mathbf{L}$, $\mathbf{B}$ and $\mathbf{B}^T\mathbf{L}\mathbf{B}$ in Equation 8.

$\mathbf{B}^T\mathbf{D}\mathbf{B}$ takes only $O(M^2N)$ time because $\mathbf{D}$ is diagonal. Solving the real non-symmetric generalized eigensystem takes $O(N^3)$ time, where the size M of the training set is not relevant. Altogether, this makes $O(M^2N + MN^2 + N^3)$ time for setting up and solving Equation 8.

3.2 Adaptive Grid Refinement

The number of grid points should be small in order to keep computation times short. At the same time, however, it might be necessary to increase the resolution of the sparse grid. Adaptive grid refinement balances between these two contrary objectives by spending more grid points only in regions where it seems necessary.

This section discusses the impact of the grid resolution on the clustering results, describes adaptive grid refinement based on hierarchical surplus, and introduces methods for adding new training instances after refining the grid.

The spatial proximity of pixels influences their assignment to clusters for at least three reasons. First, nearby pixels are likely to originate from the same object in the image, and are likely to have similar appearance, which might be reflected in the computed histogram features. Second, when computing the local color histograms, the windows overlap for nearby pixels. Third, the resolution of the sparse grid induces a smoothness constraint on the score variables for nearby pixels.

Hierarchical surplus refinement chooses to refine those grid points with the largest absolute coefficients $|\alpha_i|$, where α_i is also called hierarchical surplus. In

this work, Equation 8 is solved multiple times. In each iteration, the grid is refined at grid points with maximum surplus. The information gained at grids with fewer grid points is exploited for the next iteration.

Adding grid points introduces additional degrees of freedom. It seems natural to also include more training data after refining the grid such that these freedoms can be determined with enough support from the training data. We propose three variants of adding new training instances after refining the grid, and will later see whether this intuitive idea is successful in practice. The variants are sketched in Figure 7.

It is common to all three proposed variants that, after solving Equation 8 in the i-th iteration, we refine the grid, thereby introducing n_i new grid points. Then, for each new grid point, we sample a constant number c of pixels from a certain region. The $c \cdot n_i$ pixels in total are added to the training set, which influences the next iteration.

The first variant is the simplest of all three: for each new grid point we sample $c \cdot n_i$ pixels randomly from the whole image. The second variant samples c pixels for each new grid point within the support of its basis function. The third variant samples c pixels in a circle of fixed radius around each new grid point.

Figure 8 sketches the pipeline with all the stages required to produce the segmentation image from the input image. Except for the stage where new training points are added, an implementation has been provided by Benjamin Peherstorfer.

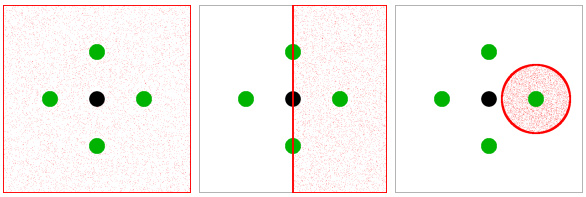

Figure 7: An illustration of the three variants for adding new training points, assuming a zero-boundary regular grid that consists of only one grid point before refinement. After refining this grid point, four new grid points (green) are added. For the new grid point to the right, the figure shows the newly added training pixels as freckles. The additional training pixels for the three other new grid points are not shown.

3.3 Model Selection

Some parameters of the algorithm have a large impact on the results. Even more, for some parameters, good parameter values vary from image to image. If this is the case, such as with the number of clusters k and the kernel parameter σ, then automatic model selection can help in choosing parameters for a given image.

As a model selection criterion, we adopt the *balanced linefit* [4] in order to estimate the goodness of clustering on a validation set. In this application scenario, the validation set is formed by pixels sampled from the image.

The balanced linefit is a linear combination of two measures; the *balance* is the ratio between the sizes of the smallest and the largest cluster. The *linefit* measures collinearity of the score variables for each cluster.

$$\text{blf} = \eta \,\text{linefit} + (1 - \eta)\,\text{balance} \tag{9}$$

The balance is computed from the clusters $\{\mathcal{A}_1, ..., \mathcal{A}_k\}$ in the validation data.

$$\text{balance} = \frac{\min_p |\mathcal{A}_p|}{\max_p |\mathcal{A}_p|} \tag{10}$$

Let $Z_p \in \mathbb{R}^{|\mathcal{A}_p| \times (k-1)}$ be the centered validation scores for the p-th cluster. Then the principal eigenvalues $\tau_{p,i}$ of the centered validation scores Z_p capture the amount of variance in the principal components. For $k > 2$

$$\text{linefit} = \frac{1}{k} \sum_{p=1}^{k} \frac{k-2}{k-1} \left(\frac{\tau_{p,1}}{\sum_i \tau_{p,i}} - \frac{1}{k-1} \right) \tag{11}$$

The case $k = 2$ requires special treatment. Although Alzate and Suykens [4] show how to do this with weighted kernel principal components analysis,

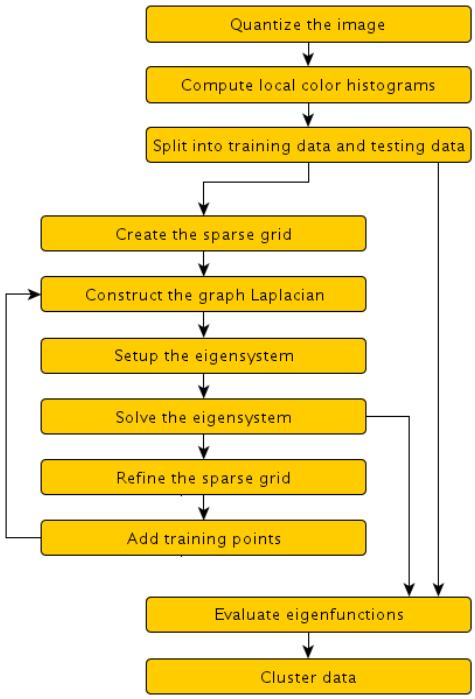

Figure 8: A schematic overview of the clustering pipeline.

it is not clear how to transfer the extension for $k = 2$ to the context of the discussed algorithm.

In principle, the eigenvalues for Equation 11 can be obtained by solving of $\frac{1}{|\mathcal{A}_p|} Z_p Z_p^T \mathbf{v} = \tau_{p,i} \mathbf{v}$. However, in our implementation, we compute the singular value decomposition of the centered validation scores Z_p instead, using the `gsl_linalg_SV_decomp_mod` routine in the GNU Scientific Library. Thus, the sample covariance matrix does not need to be computed.

The squares of the singular values of Z_p are equal to the eigenvalues of the covariance matrix $\frac{1}{|\mathcal{A}_p|} Z_p Z_p^T$, but only up to scale. The scale does not matter in Equation 11, as it is cancelled out in the ratio.

The boundary case, in which the sample validation data does not contain any instances assigned to a particular cluster p, needs to be treated as well in the implementation. Alzate and Suykens [4] do not specify this case, so we suggest to just set the p-th summand in Equation 11 to zero, thereby indicating that a clustering with "empty" clusters is no good.

In order to obtain good parameters for a particular test image through automatic model selection, we can cluster the data with different parameter values, and then select the clustering with the highest balanced linefit. This means that all stages in the algorithm that depend on these parameters need to be re-run.

3.4 Cluster Evaluation

It is useful to be able to quantitatively assess the results of the segmentation algorithm. The Berkeley Segmentation Dataset 300 ships with code[2] that quantitatively compares computed segmentation boundaries with boundaries marked by humans, in the following referred to as ground truth. The ground truth for a single image consists of a probability boundary map, which specifies the probability for each pixel that it belongs to a boundary. Precision and recall are calculated from the computed boundary map and the ground truth boundary map. The final reported quantity is the *F-measure*, which is the harmonic mean of precision and recall. A graph of the F-measure versus precision and recall is shown in Figure 9.

3.5 Experiment Runner

Any machine learning algorithm requires the selection of suitable parameter values. In the discussed algorithm, the number of algorithm parameters is quite large. The dimensionality of the parameter space can be reduced by fixing the values of parameters that supposedly have low impact. Instead, we concentrate on choosing good values for parameters with supposedly high impact on the results. We implemented an experiment runner that helps understand the impact of parameters.

The experiment runner is fairly basic. While grid search is desirable, it is computationally expensive, and an extensive quantitative analysis is mostly out of scope in this work. The experiment runner supports

[2] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds

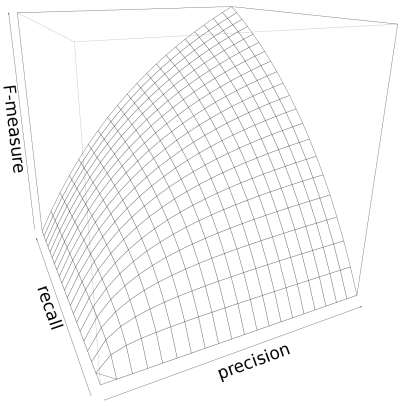

Figure 9: A 3-dimensional plot of the F-measure versus precision and recall.

testing different values for a single parameter at a time. This is relevant to the evaluation in Section 4, because many of the presented results are reported on the margins of the parameter space.

However, the experiment runner is well-suited for obtaining preliminary qualitative results. It is well-equipped for the visual inspection of segmented images. Input, output, and various statistics are collected in a principled manner in a database.

4 Evaluation

This section discusses the results of the clustering algorithm. The main purpose is to gain a better understanding of the new clustering approach with eigenfunctions learned on a sparse grid. Effects of the grid level are described in Section 4.4. The methods for adaptive grid refinement, including those introduced in this report, are treated in Section 4.5 and Section 4.6. Results on the Berkeley Segmentation Dataset are given in Section 4.8. The balanced linefit criterion for model selection is subject of Section 4.7. Finally, the performance is discussed in Section 4.10. The primary goal is to obtain qualitative results.

4.1 Data for Evaluation

We use synthetic toy examples in order to gain insights because these simple data make reasoning easier. In parallel, the algorithm is evaluated on five test images from the Berkeley Segmentation Dataset 300 (BSDS300). It is important to note that we are *not* comparing our results on these five images to other methods evaluated on these test images of the BSDS300. It is more like asking: given the test images, how far can we take the system to produce good results, without introducing a too large bias? Answers to this question, together with the performance published in related work on the test images, might indicate in which directions to go in future work.

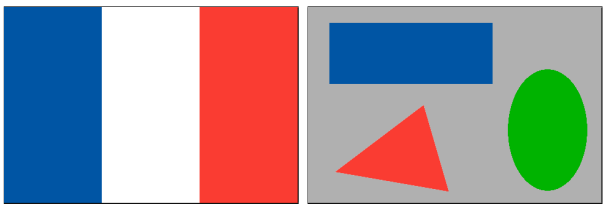

Figure 11: Two synthetic toy images (the borders are not part of the data).

The toy examples comprise the *tricolor* image (Figure 11, left), and the *shapes* image (Figure 11, right). The *tricolor* image exhibits discontinuities in one direction, while the *shapes* image shows discontinuities in various directions. The perfect segmentation can be directly seen on these toy images.

For more complex data, we consider the images *elephant*, *bird*, *pyramids*, *surfer*, and *parade* (Figure 10, from left to right, from top to bottom) in the BSDS300.

4.2 Visual Assessment

Throughout Section 4, we often blend the original image, the segmentation image, a plot of the sparse grid, and a plot of the training set into a single image in order to make it easier to assess the results visually. For example, on any of the images in Figure 12, one can recognize the original image faintly in the background. The clusters are drawn in semi-opaque

Figure 10: Five test images from the Berkeley Segmentation Dataset 300.

colors, the training points are drawn as freckles, and the grid points as small black crosses.

4.3 Parameters for Evaluation

The clustering algorithm described in Section 3 has several parameters. Table 1 lists all parameters that are fixed for all experiments. All images have the same size. The width and the height of the window used for computing the local color histograms is fixed. The pixels at the margin, where the histograms cannot be computed without padding the image, are discarded in the implementation. The histograms for the images from the BSDS300 all have eight bins, whereas those of *tricolor* have three bins, and those of *shapes* have four bins. The initial training set is always sampled randomly from the image without replacement. The size of the training set varies, but is usually chosen to be less than 6% of all pixels. All experiments are conducted with a linear boundary grid, using the routine `Grid::createLinearBoundaryGrid` in the `SG++` framework. The regularization parameter λ is fixed. Table 2 show the varying parameters. Table 7 lists the parameter values for all experiments referenced in this report. Note that n_α equals k in some experiments. This is for historical reasons.

Parameter	Value
image width	481
image height	321
window size	5
grid boundary	linear
regularization λ	0.1

Table 1: Parameters that are the same for all experiments presented in the evaluation.

Parameter	Variable
kernel parameter	σ
number of clusters	k
considered eigenvectors	n_α
initial grid level	l
refinement steps	n_r
refinement percentage	$r_\%$
additional training instances	c
maximum refinement level	$r_{\max}$

Table 2: Parameters that vary for the experiments presented in the evaluation.

4.4 Different Grid Levels

One objective of this work is to achieve good results with few grid points. First, we test regular sparse grids with linear boundaries at different grid levels in order to see what level is sufficient to obtain good segmentations without refinement.

Figure 12 shows the results for images *elephant*, *bird*, and *surfer*. The images were tested at regular grids ranging from levels 3 to 8. The results indicate that a regular grid of level 7 is appropriate for these images because the results for levels 6 and 7 differ widely, while there is little visible change between levels 7 and 8.

Table 3 lists the number of grid points for each level of grids with linear boundaries. For the levels listed, the grid size at least doubles with each level. Hence, the selection of a good level can save vast computational efforts. Note that the reported levels and grid sizes are consistent with the levels used in

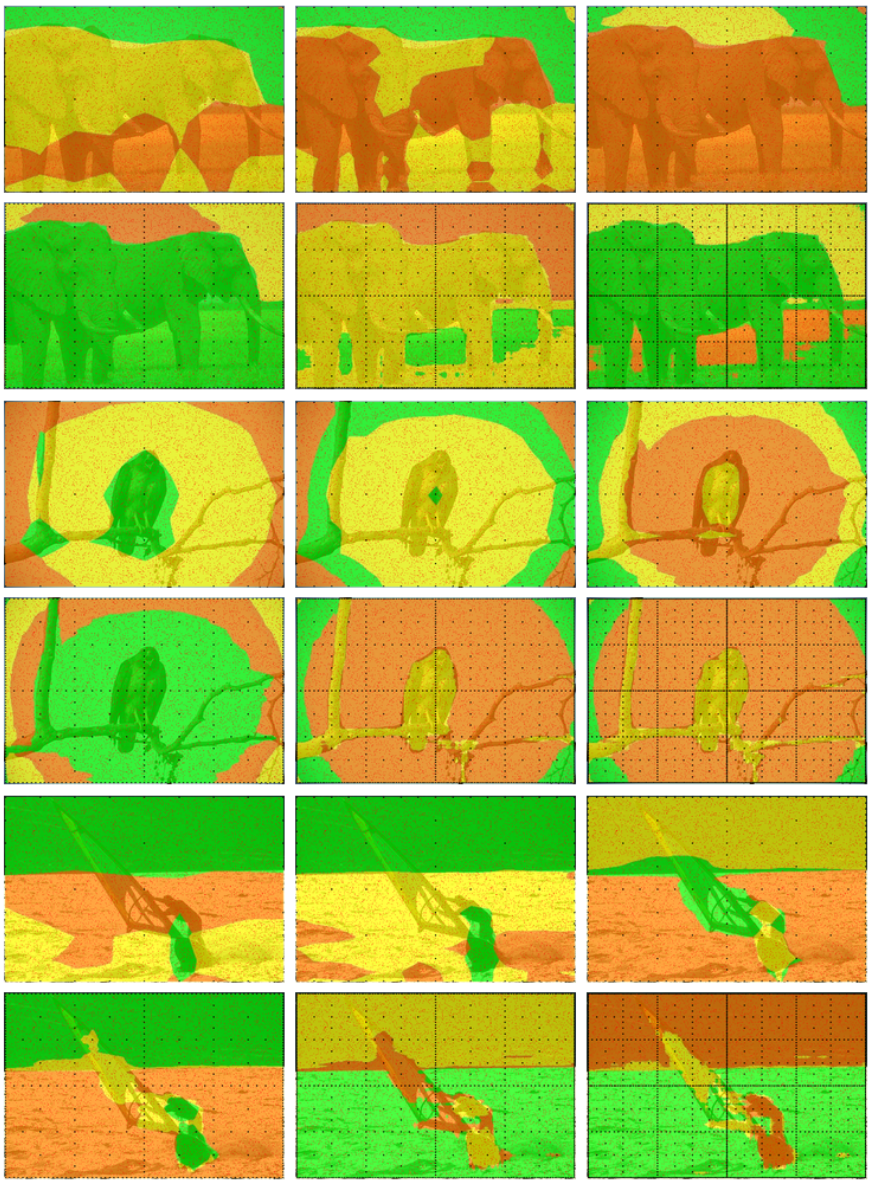

Figure 12: Segmentation results at grid levels ranging from 3 to 8 for the images *elephant*, *bird*, and *surfer*.

grid level	2	3	4	5	6	7	8	9
inner points	1	5	17	49	129	321	769	1793
boundary points	16	32	64	128	256	512	1024	2048
grid points	17	37	81	177	385	833	1793	3841

Table 3: Number of grid points for regular grids with linear boundaries at different levels.

the `SG++` framework.

4.5 Adaptive Refinement Results

This section shows results of the adaptive refinement strategies, which have been described in Section 3.2. We first look at the hierarchical surplus refinement alone, and then move on to the extensions where training points are successively added in each iteration.

For a beginning, we show the effects of hierarchical surplus refinement on the *tricolor* image, because this toy example shows clear segmentation boundaries that should correspond to sharp edges in the learned eigenfunctions. We start with a level $l = 2$ grid and refine $n_r = 3$ times. Figure 13 (top) shows the segmentation result. As expected, the algorithm segments the simple image perfectly. The grid has high resolution at the true segmentation boundaries. Notably, the refined grid points increase the resolution in the second dimension.

When the size of the training set is too low compared to the number of grid points, then overfitting occurs. Figure 13 (bottom) shows an example for such overfitting. It indicates that high resolution should be backed by a large enough training set.

Next, we look at the *elephant*, the *bird*, and the *surfer* in order to check for patterns in the refined grids. The initial grid is a regular level 4 grid, and 10% of all grid points are refined after each step. Figure 14 shows the segmentation results for the *elephant* (left) after five refinement steps, and the results for both the *bird* (center) and the *surfer* (right) after six refinement steps. The resolution is high in areas which correspond to computed segmentation boundaries. Conversely the grid resolution is low in areas where the pixels have been assigned to the same cluster. In this case, the areas of high resolution cover the true boundaries of the objects in the image.

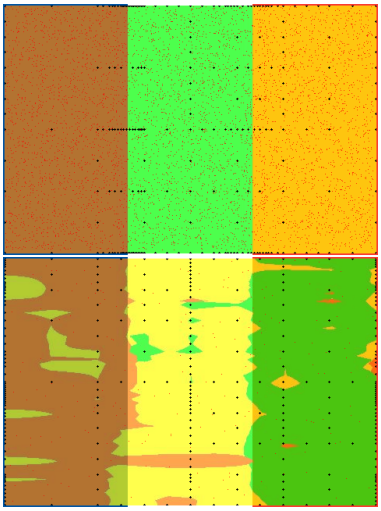

Figure 13: Segmentation results for the *tricolor* image with adaptive refinement.

Finally, we compare the number of grid points in a refined grid and a comparable regular grid. Figure 15 shows the result of an experiment with the *elephant*, the *bird*, and the *surfer*. In the left two columns, results on regular grids at level 6 and level 7 are given. The right column shows qualitatively similar results on refined grids. Table 4 lists the number of grid points on the refined grids. Both the regular grids at level 7 and the refined grids lead to better segmentations than the regular grid at level 6. However, the refined grids require about 350 fewer grid points than the level 7 grid in order to achieve the same improvement over the level 6 grid.

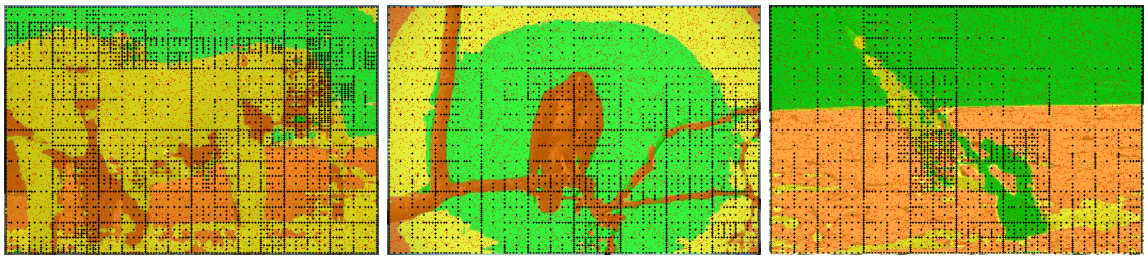

Figure 14: Segmentation results for the *elephant* image (left), the *bird* image (center), and the *surfer* image (right) on excessively refined grids.

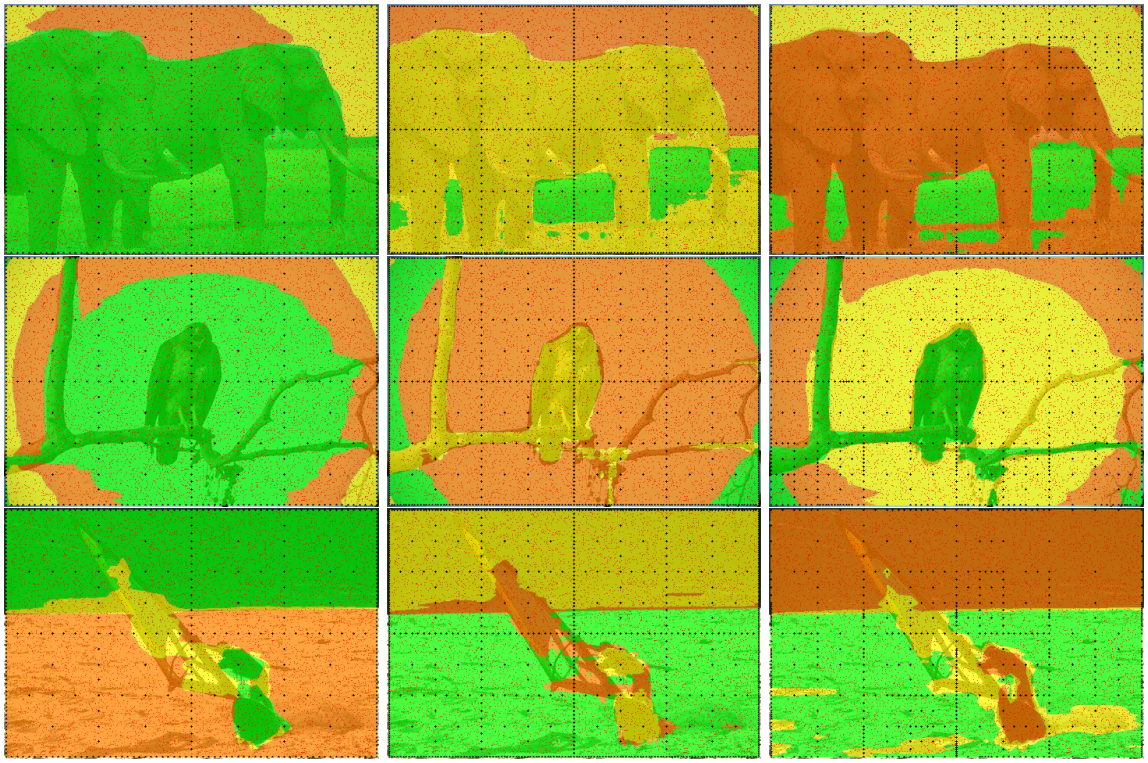

Figure 15: Segmentation results on a regular grid of level 6 (left column), on a regular grid of level 7 (center column), and on an adaptively refined grid with fewer grid points than a regular grid of level 7 (right column).

image	refined grid	refinement steps
elephant	499	2
bird	497	3
surfer	514	3

Table 4: Listing of the number of grid points for the images in the right column of Figure 15.

4.6 Additional Training Instances

This section evaluates the three variants of the algorithm which add new training instances (see Section 3.2). Adding training points only within the support of the basis functions around new grid points turned out to degrade the results. The problem is apparent in Figure 16 (top). The refinement only happens in one quadrant of the image. The algorithm gets trapped. This is undesirable. The same effect is observable in the tricolor image presented in Figure 16 (bottom). While there is not enough evidence to fully discard the idea, it is an unpromising path.

Adding training points randomly from the whole image is more promising. The top image in Figure 13 has been created with only 200 training instances in the first iteration. After refining, for each newly added grid point, 60 new training instances have been added until reaching a final number of 5780 training points.

4.7 Model Selection Results

This section presents the results of using the balanced linefit criterion to automatically select the number of clusters and the kernel parameter.

We set $\eta = 0.75$ in Equation 9, as selected by Alzate and Suykens [4]. First, we let the number of clusters k run from 3 to 6, and the kernel parameter σ run from 0.05 to 1.00 with a step of 0.05. The other algorithm parameters were kept fixed. Table 5 shows the chosen pair (k, σ) for each of the images.

Figure 17 shows the balanced linefit plotted against k and σ for all three images in Table 5. It appears that the balanced linefit prefers very few clusters in this context. The same parameter values as in Table 5 would have been selected based on linefit only (i.e. with $\eta = 1$). Figure 18 shows the selected segmentation images.

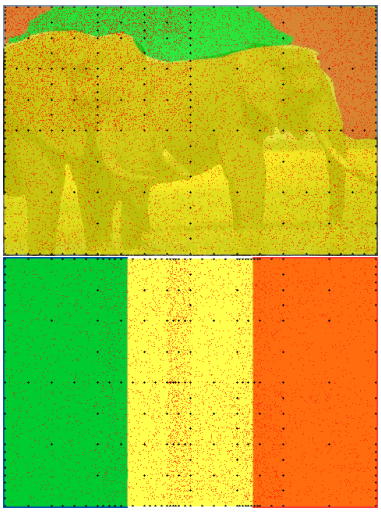

Figure 16: Segmentation results for the *elephant* image (top), and the *tricolor* image (bottom). These results indicate that adding training points in the support area of new grid points is counterproductive.

image	k	σ	*lf*	b	*blf*
shapes	3	0.75	0.89	0.18	0.71
elephant	4	0.15	0.77	0.15	0.61
parade	3	0.05	0.86	0.66	0.81

Table 5: Model selection results: number of clusters k, kernel parameter σ, linefit *lf*, balance b, balanced linefit *blf*.

4.8 BSDS300 Benchmark

This section analyzes the segmentation results quantitatively. The algorithm is evaluated on 20 randomly selected images out of the 100 test images in the BSDS300 dataset. The parameters σ and k are chosen by the balanced linefit criterion, with levels $0.10, 0.55, 1.00$ for σ and levels $3, 4, 5$ for k. The initial grid level is set to 7. The grid is refined once

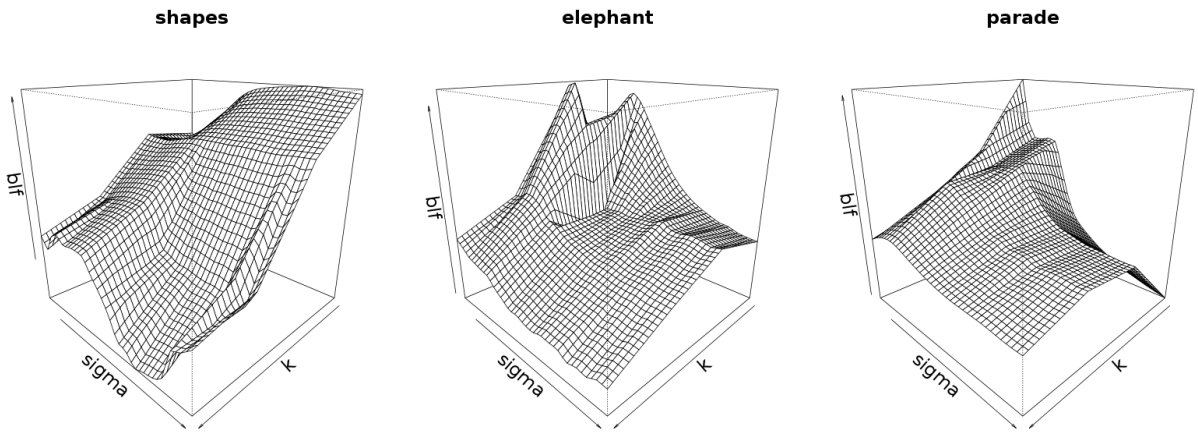

Figure 17: Plots of the balanced linefit versus the number of clusters and the kernel parameter for the images *shapes*, *elephant*, and *parade*. The computed values of the balanced linefit are interpolated for better visualization.

Figure 18: Segmentation results for the images *shapes* (left), *elephant* (center), and *parade* (right). These results were obtained with the parameters chosen by maximum balanced linefit.

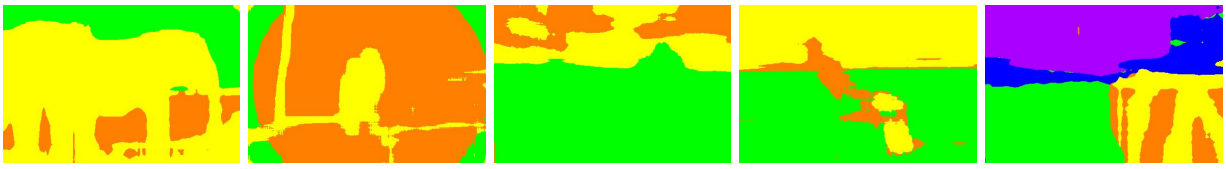

Figure 19: Hand-picked segmentation results for five images from the Berkeley Segmentation Dataset 300.

Image ID	σ	k	F (a)	F (b)
8023	0.10	4	0.21	**0.31**
14037	0.55	3	0.46	**0.58**
33039	0.10	3	**0.56**	0.48
38092	0.10	3	0.54	**0.62**
41033	0.10	3	0.51	**0.64**
76053	0.10	3	0.23	**0.59**
85048	0.10	3	0.61	**0.62**
97033	0.55	3	**0.54**	0.51
101087	0.10	3	0.31	**0.60**
108082	0.10	3	0.35	**0.36**
109053	0.10	3	0.31	**0.48**
134035	0.10	3	0.45	**0.48**
160068	0.55	3	0.43	**0.50**
189080	1.00	3	0.64	**0.65**
208001	0.10	3	0.34	**0.60**
236037	0.10	3	0.26	**0.48**
291000	0.10	3	0.42	**0.53**
296007	0.10	3	0.41	**0.66**
299086	0.10	4	0.50	**0.70**
306005	0.55	3	0.45	**0.63**

Table 6: F-measures (a) for this work, and (b) for the work by Alzate and Suykens [4].

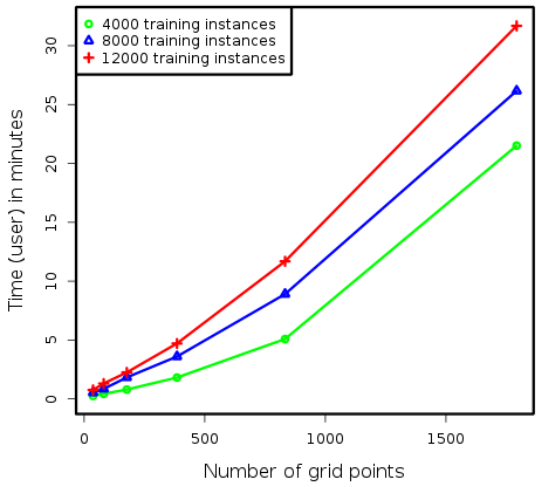

Figure 20: CPU time versus the number of grid points at three different sizes of the training set.

with $r_{\%} = 0.03$. Table 6 shows the results. Alzate and Suykens [4] achieve a larger F-measure on most images.

4.9 Selected Results

In order to show some promising results, we hand-picked five segmentation images. These are shown in Figure 19. The selected results show that the algorithm is able to properly segment the five images *elephant*, *bird*, *pyramids*, *surfer*, and *parade*. Of course, part of the success is due to the manual selection of the best segmentations. However, these results show what is possible to achieve if the right model is chosen.

4.10 Performance

This section presents the computation times of the algorithm. The computation times depend heavily on the chosen parameter values. While some stages in the algorithm support parallel execution, this evaluation is concerned with *user* time rather than wall time. The times are measured on a notebook with an Intel i5 quad-core processor at 2.67 GHz.

We first examine the impact of segmenting images at grid levels ranging from 3 to 8. Figure 20 shows the CPU time versus the number of grid points. The times for the six different grid levels are linearly interpolated for better visualization. Times are reported for 4000, 8000, and 12000 training instances. The computation times grow non-linearly with the number of grid points, which is consistent with the analysis in Section 3.1.

A second experiment shows the relationship between the number of training instances and the computation time. We tested configurations with up to 16000 training instances at three different grid levels. Figure 21 shows the results. The graph indicates that the cost for adding training instances is higher on a fine grid than on a coarse grid. This is noteworthy, but also in line with the theoretical analysis in Section 3.1.

Fig.	σ	k	n_α	l	M	n_r	c	$r_{\%}$	$r_{\max}$
12.1	0.30	3	3	3	8000	0	25	-	-
12.2	0.30	3	3	4	8000	0	25	-	-
12.3	0.30	3	3	5	8000	0	25	-	-
12.4	0.30	3	3	6	8000	0	25	-	-
12.5	0.30	3	3	7	8000	0	25	-	-
12.6	0.30	3	3	8	8000	0	25	-	-
12.7	0.30	3	3	3	8000	0	25	-	-
12.8	0.30	3	3	4	8000	0	25	-	-
12.9	0.30	3	3	5	8000	0	25	-	-
12.10	0.30	3	3	6	8000	0	25	-	-
12.11	0.30	3	3	7	8000	0	25	-	-
12.12	0.30	3	3	8	8000	0	25	-	-
12.13	0.30	3	3	3	8000	0	25	-	-
12.14	0.30	3	3	4	8000	0	25	-	-
12.15	0.30	3	3	5	8000	0	25	-	-
12.16	0.30	3	3	6	8000	0	25	-	-
12.17	0.30	3	3	7	8000	0	25	-	-
12.18	0.30	3	3	8	8000	0	25	-	-
13.1	0.45	3	3	2	200	3	60	0.10	∞
13.2	0.20	3	2	2	200	5	25	0.30	7
14.1	0.50	3	3	4	8000	5	25	0.10	7
14.2	0.50	3	2	4	8000	6	25	0.10	7
14.3	0.50	3	2	4	8000	6	25	0.10	7
15.1	0.30	3	3	6	8000	0	25	-	-
15.2	0.30	3	3	7	8000	0	25	-	-
15.3	0.50	3	3	4	8000	2	25	0.10	7
15.4	0.30	3	3	6	8000	0	25	-	-
15.5	0.30	3	3	7	8000	0	25	-	-
15.6	0.50	3	2	4	8000	3	25	0.10	7
15.7	0.30	3	3	6	8000	0	25	-	-
15.8	0.30	3	3	7	8000	0	25	-	-
15.9	0.50	3	2	4	8000	3	25	0.10	7
16.1	0.45	3	3	2	4000	3	80	0.10	∞
16.2	0.45	3	3	2	200	3	60	0.10	∞
18.1	0.05	3	2	7	8000	0	25	-	-
18.2	0.05	3	2	7	8000	0	25	-	-
18.3	0.05	3	2	7	8000	0	25	-	-
19.1	0.45	3	3	3	4000	3	40	0.10	∞
19.2	0.30	3	3	8	8000	0	25	-	-
19.3	0.35	3	3	7	8000	0	25	-	-
19.4	0.30	3	3	7	8000	0	25	-	-
19.5	0.35	6	5	6	4000	0	25	-	-

Table 7: Parameters for all experiments referenced in this report. M is the initial size of the training set.

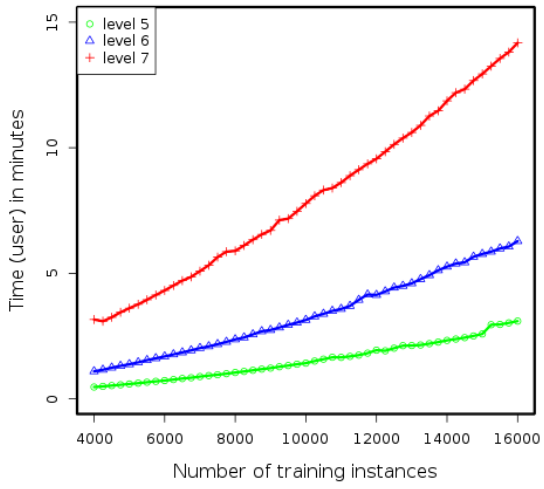

Figure 21: CPU time versus the number of training instances at three different grid levels.

5 Conclusion

5.1 Contributions

This work consists of several parts. We extended the hierarchical surplus refinement strategy such that new training instances can be added in each iteration. We transferred the balanced linefit criterion to the algorithm described in this work. We created tools for visualizing intermediate results and the final segmentation results. We implemented an experiment runner which is able to store results in a database for convenient access. We examined the impact of different algorithm parameters, and showed that good segmentation results can be achieved with few grid points.

5.2 Results

In this report, we list both insuccessful attempts and promising results. The non-linear clustering method from Peherstorfer et al. [2] can be applied to image segmentation. It is especially promising in combination with adaptive refinement based on hierarchical surplus. This refinement strategy tends to increase the resolution at object boundaries and segmentation boundaries. Adaptive refinement allows us to obtain segmentation results of the same quality as with regular sparse grids, but involves fewer grid points.

The concept of adding training instances in each iteration is sound, but confining the sampling area to the support of basis functions is counterproductive. The resolution of the sparse grid and the local spatial consistency are tied together. Greater level of detail can be obtained by increasing the resolution of the grid.

The balanced linefit criterion seems to prefer few clusters in the tested images. As a subjective human observer, the criterion chooses too few clusters, and it needs to be looked at more closely.

Good segmentation results can be obtained in around 8 minutes of CPU time. The required computation times depend mainly on the number of iterations, the number of grid points, and the number of training instances.

5.3 Future Work

Many questions remain open for future work. This applies to feature extraction, to model selection, to quantitative evaluation, and to performance.

This work used features exclusively based on local color histograms. Future work might include other types of features, such as explicit spatial cues.

An objective, quantitative criterion such as the F-measure is very helpful for evaluation. While benchmark results on the BSDS300 are given in this report, the quantitative analysis has not been focus of this work. A well-integrated quantitative criterion would help in finding an optimal tradeoff between initial grid level and number of refinement steps in terms of computation time and segmentation quality.

Model selection has not been the main focus in this work. Besides of the balanced linefit criterion, other model selection mechanisms could be tried in future work. In order to assess the results, a quantitative criterion for evaluation would be helpful here as well.

The computation times can be reduced through parallelization. A remaining task is to find out which

of the available solvers for the particular eigensystem in Equation 8 is the fastest in practice.

References

[1] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[2] B. Peherstorfer, D. Pflüger, and H.-J. Bungartz, "A Sparse-Grid-Based Out-of-Sample Extension for Dimensionality Reduction and Clustering with Laplacian Eigenmaps," in *AI 2011: Advances in Artificial Intelligence*, vol. 7106 of *Lecture Notes in Computer Science*, pp. 112–121, Springer, 2011.

[3] M. Belkin and P. Niyogi, "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[4] C. Alzate and J. A. K. Suykens, "Multiway Spectral Clustering with Out-of-Sample Extensions through Weighted Kernel PCA," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 2, pp. 335–47, 2010.

[5] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral Grouping Using the Nystrom Method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 214–225, 2004.

[6] P. Heckbert, "Color Image Quantization for Frame Buffer Display," *Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques*, vol. 16, no. 3, pp. 297–307, 1982.

[7] X. Wu, "Efficient Statistical Computations for Optimal Color Quantization," in *Graphics Gems II* (J. Arvo, ed.), pp. 126–133, Academic Press, 1991.

[8] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics," in *Proceedings of the 2001 IEEE International Conference on Computer Vision*, vol. 2, pp. 416–423, 2001.

[9] U. von Luxburg, "A Tutorial on Spectral Clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[10] H.-J. Bungartz and M. Griebel, "Sparse grids," *Acta Numerica*, vol. 13, pp. 147–269, 2004.

[11] T. Gerstner and M. Griebel, "Sparse grids (short introduction)," in *Encyclopedia of Quantitative Finance*, John Wiley & Sons, 2008.