Clustering methods

Final report

SORIANO Tristan
AUBENEAU Simon
DUVAL Quentin
PRIEUL Simon
SANTINA Jérémy

March 24, 2015

Contents

Special thanks

The team would like to thank Laurent Beugnet, Ronan Guivarch and Sandrine Mouysset for their advices,the informations and time they gave us and patience they showed. It was a real pleasure to be part of this project in their company.

Introduction

Introduction

In image processing pattern recognition tend to find similarity and regularity into a dataset. It can be used in different domain such as medicine like visualizing pattern and shape on an echography, research: analyzing thermal exchange in a gas environment or even security following object on a video. The entry can be a geometrical image or a color image. It is composed of many points or pixels that must be analyzed in a way we find the property they share. As the entry dataset can represent a huge matrix it is crucial to find an efficient algorithm to minimize calculus.

Problem statement

As mentioned previously, this is a time and memory consuming process. To optimize that, it has been decided to parallelize the code. A master process splits the dataset into chunk and spread them to slaves processes. Each slave processes the dataset to find cluster then share its results to the master that merges them. The objective was to enhance and adapt an existing code to allow the use of new clustering methods into a such master slave structure. There are various way to label an input using different algorithm. The existing code use a method called spectral clustering. It projects the existing dataset into a feature space of higher dimension to spread the point into the space. This procedure enables to separate point using more precise hyper plans. The algorithm builds a similarity matrix and extract the point are share the same eigen sub space. The eigen value computation is really heavy and it highlighted the interest of using lighter methods. Approach Our work had different goals. The first on was about existing code refactoring. We had to modify an existing research code to improve its portability on different compilers. Then, we had to internationalize it and create a documentation for other researcher that would have to work with it. Finally we had to produce an interface for new method implementation and enable the use of different kernel algorithm.

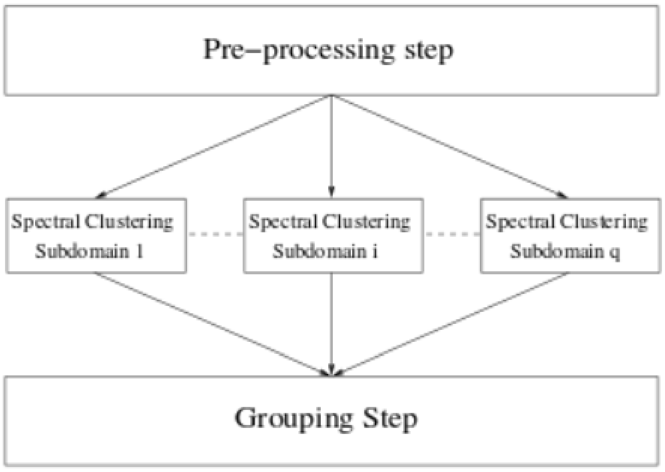

Pre−processing step
Spectral Clustering Subdomain 1
Spectral Clustering Subdomain i
Spectral Clustering Subdomain q
Grouping Step

Figure 1

Documentation

The following reference can be used for any theoretical understanding of the algorithm. It also provides references on the minimum interface the different method must follow.

- The Global Kernel k-Means Clustering Algorithm Grigorios Tzortzis and Aristidis Likas
- The Variable Bandwidth Mean Shift and Data-Driven Scale Selection Dorin Comaniciu Visvanathan Ramesh Peter Meer
- Mean Shift Analysis and Applications Dorin Comaniciu Peter Meer

Part I

Project Management

Chapter 1

Development plan

1.1 Development plan

This project had different purposes. First the goal was to clean the existing code, then to refactore it has much has possible: in different way internationalizing it, making it more reusable and so on. Finally, adding new clustering methods into the existing code and finding way to make it more flexible to new methods implementation through new interfaces. The client had an existing development plan in mind that was logical, covered all its demands and that we decided to follow.

1.1.1 Work breakdown structure

Each of the following step represents the schedule for the different point expected by the client to be complete. It check the good evolution of the project and the respect of the desired specifications.

Step 1

- 1. Bibliography study (the reference can be found in the documentation section)
- 2. Existing parallel code set up on lab machines, new example creation : 2D examples 3D examples
- 3. The three following methods must be implemented in matlab : spectral clustering, Kernel K-Means and Mean Shift

Step 2

- 1. Code documentation (Doxygen) : dependency graphs, method and parameters description.
- 2. End of the clustering methods implementation in Matlab

Step 3

- This step provides specification of the new clustering methods interfaces in Fortran and validation with the client.

Step 4

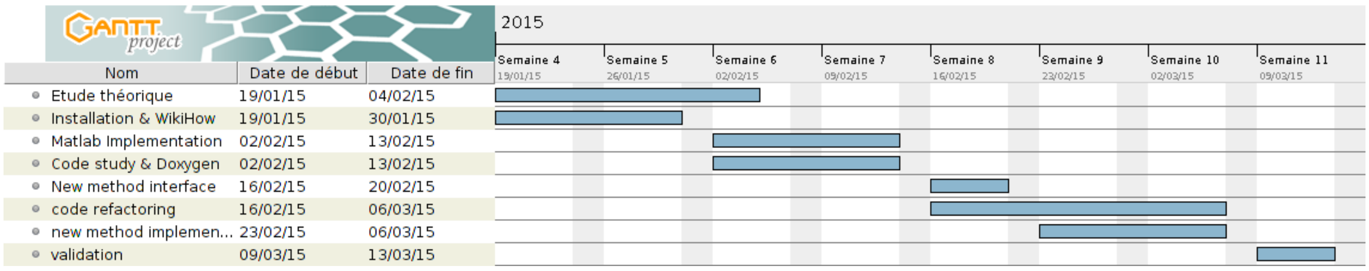
- 1. Code refactoring : the code will follow classical Fortran coding convention, the method and variables will be renamed for better understanding.
- 2. Implementation of the new clustering methods interfaces FORTRAN
- 3. new tests generation

Step 5

- Validation of the new methods. Quality of the result on the different tests, time elapsed computing, non regression check.
- The refactored code will be tested and the validation will rely on statical analysis of the code.

We followed this development plan.However with a such structure we tend to think that the refactoring and documentation part would be fast. But, as explained later it asks to work in parallel with implementation and documentation. Giving this parallel work issue, the task ended at the same time as the project and the step 2 and 4 only indicates the beginning of the task.

We ended up with the following initial gantt chart :



Nom	Date de début	Date de fin
Etude théorique	19/01/15	04/02/15
Installation & WikiHow	19/01/15	30/01/15
Matlab Implementation	02/02/15	13/02/15
Code study & Doxygen	02/02/15	13/02/15
New method interface	16/02/15	20/02/15
code refactoring	16/02/15	06/03/15
new method implemen...	23/02/15	06/03/15
validation	09/03/15	13/03/15

Figure 1.1: *Initial gantt*

Chapter 2

Specification plan

2.1 Overview

We define here the expectation of the client and the corresponding specification. The developer will follow the needs specified by the customer as long as its part of the following agreement. This is a research project on image processing. It produces clustering creation among point sets and images. A such treatment can be applied on real time identification and following on a video.

2.2 Documentation

Here is a useful documentation provided by the client in order to understand the three clustering algorithms: Spectral Clustering (already implemented), Kernel K-Means, Mean-Shift and how the first one have been implemented. This should serve for algorithm implementation and overall comprehension.

2.2.1 Applicable Documents

Date	**Title**	**Author(s)**
20 Jan. 2009	The Global Kernel k-Means Clustering Algorithm (revised version)	Grigorios Tzortzis Aristidis Likas
2001	The Variable Bandwidth Mean Shift and Data-Driven Scale Selection	Dorin Comaniciu Visvanathan Ramesh Peter Meer
unknown	Mean Shift Analysis and Applications	Dorin Comaniciu Peter Meer

2.2.2 Reference Documents

Date	**Title**	**Author(s)**
unknown	On Spectral Clustering: Analysis and an algorithm	Andrew Y. Ng Michael I. Jordan Yair Weiss
22-25 Aug. 2004	Kernel k-means, Spectral Clustering and Normalized Cuts	Inderjit S. Dhilon Yugjang Guan Brian Kulis

2.3 Deliverables

Here are the expected deliverables that the client will receive at the end of the project.

Content
Implemented Kernel K-means and Mean-Shift algorithms
Running configuration files with documentation
Refactored code from the given sources to improve maintainability and optimized compilation
Documentation of the source code in HTML and PDF formats
New tests for comparison of performances between algorithms

2.4 Functional Requirements

Here are the new functionalities that the program should implement.

Reference	Requirement
FREQ-01	Kernel K-means should work with all different data format : 2D or 3D coordinates, picture, thresholded picture and geometric (picture + coordinates)
FREQ-02	Mean-Shift should work with all different data format : 2D or 3D coordinates, picture, thresholded picture and geometric (picture + coordinates)
FREQ-03	The user should be able to use any of the three algorithms using a new parameter defined in the *param.in* file.
FREQ-04	The user should be able to set a bandwidth for Mean-Shift algorithm using a new parameter defined in the *param.in* file.
FREQ-05	New data sets should be created to provide other tests for the algorithms.

2.5 Non-Functional Requirements

Here are the requirements that do not implement new functionalities for the program. They concern the uptime requirements, the reverse engineering aspect and the maintainability of the code.

Reference	Requirement
NFREQ-01	The deadline for this project is the 13th of March 2015.
NFREQ-02	The source code should be documented using specific comments inside the code.
NFREQ-03	The deliverables should be put on versionned repository using Git.
NFREQ-04	Quality of the original source code should be improved by removing unused variables and methods and by replacing deprecated types and symbols.
NFREQ-05-1	Readability of the original source code should be improved by renaming variables, methods and parameters using proper naming convention.
NFREQ-05-2	Readability of the original source code should be improved by translating all French portions of the code into English.
NFREQ-05-3	Readability of the original source code should be improved by removing all commented code.
NFREQ-06	The documentation should be written in English.
NFREQ-07	The algorithms should be as efficient as possible in term of performances.

2.6 Environmental Requirements

Here are the requirements on the technologies that are used: software and hardware.

Reference	Requirement
ENVREQ-01	The program should work on ENSEEIHT's computer science teaching equipment.
ENVREQ-02	The new algorithms Kernel K-Means and Mean-Shift should be implemented in Fortran.
ENVREQ-03	The code should be compiled with GFortran.
ENVREQ-04	The new algorithms Kernel K-Means and Mean-Shift should be compatible with the libraries MPI, Arpack and Lapack.
ENVREQ-05	The documentation should be generated using Doxygen.

Chapter 3

Tests plan

3.1 Environment

3.1.1 Hardware environment

As it is a parallel computing program. The environment must enable the team to test on single and on multiple entity the code produced. In order to do so the code will be tested on lab machines, which are the most likely to represent the IRIT environment.

3.1.2 Software environment

This project implies the use of multiple technologies: Matlab, Fortran, MPI, ssh, Doxygen. The test must run on the client machine configuration.

3.2 Test definition

The following test must show the customer that the code fit the specifications. It includes non-regression test : the original code will be tested on the team configuration as a performance reference and will be compared to the new implemented algorithm result in term of efficiency and quality.

Id: T001

Description: This is the initial test, all the initial function of the program must run. Bouquet, 3blocSimples, Cible, arthus will be tested

Result : The program must run on 2D, 3D geometrical and color images.

- Bouquet: It must produce 4 clusters : leaf, oranges, tree, background.
- Cible: It must produce 1 cluster per ring.
- 3blocSimples: 3 clusters
- Arthus: It must run, find clusters in a limited time.

Id: T002

Description: We must test the matlab implementation of Mean Shift. Bouquet, 3blocSimples, Cible will be tested

Result: The program must run on 2D, 3D geometrical and color images.

- Bouquet: It must produce 4 clusters : leaf, oranges, tree, background.
- 3blocSimples: 3 clusters
- Cible: It must produce 1 cluster per ring.

Id: T003

Description: We must test the matlab implementation of Kernel K-Means. Bouquet, 3blocSimples, Cible will be tested

Result: The program must run on 2D, 3D geometrical and color images.

- Bouquet: It must produce 4 clusters : leaf, oranges, tree, background.
- Cible: It must produce 1 cluster per ring.
- 3blocSimples: 3 clusters

Id: T004

Description: We must test the Fortran implementation of Mean Shift. Bouquet, 3blocSimples, Cible will be tested

Result: The program must run on 2D, 3D geometrical and color images.

- Bouquet: It must produce 4 clusters : leaf, oranges, tree, background.
- Cible: It must produce 1 cluster per ring.
- 3blocSimples: 3 clusters

Id: T005

Description: We must test the Fortran implementation of Mean Shift. Arthus1 will be tested with different bandwidth

Result: The program must run on 2D, 3D geometrical and color images.

- Arthus 1 must produce an image segmentation result in limited time

---

Id: T006

Description: We must test the Fortran implementation of Kernel K-means. Bouquet, 3blocSimples, Cible will be tested:

1. using polynomial kernel function
2. using gaussian kernel function

Result: The program must run on 2D, 3D geometrical and color images.

- Bouquet: It must produce 4 clusters : leaf, oranges, tree, background.
- Cible: It must produce 1 cluster per ring.
- 3blocSimples: 3 clusters

Id: T007

Description: We must test the initial implementation of Spectral clustering using different Kernel function. Bouquet, 3blocSimples, Cible will be tested:

1. using polynomial kernel function
2. using gaussian kernel function

Result: The program must run on 2D, 3D geometrical and color images.

- Bouquet: It must produce 4 clusters : leaf, oranges, tree, background.
- Cible: It must produce 1 cluster per ring.

- 3blocSimples: 3 clusters

The test T005 o T007 will be duplicated for parallel computing

Those tests will cover:

1. Matlab implementation of Kernel K-means
2. Matlab implementation of Mean shift
3. Fortran implementation of Kernel K-means
4. Fortran implementation of Mean shift
5. Fortran implementation of Kernel methods
6. Integration of the methods into a parallel code

Id: T008

Description: We must show that the code refactoring has been successful. The code will be statically analyzed to show the improvement.

Result: The code readability, syntax must have been improved and standards must have applied.

Chapter 4

Meetings

4.1 Communication

4.1.1 External communication

The following schedule has been decided : Weekly meeting with the client (IRIT) Sandrine Mouysset, Ronan Guivarch
Weekly meeting with the supervisor Laurent Beugnet

4.1.2 Internal communication

Biweekly meeting with the team to check the evolution of the project the possible issues and change of the planning depending on the progression The text documentation can be found on Google Drive The code and the examples were shared on GitHub

Chapter 5

Feedback

5.1 Feedback

5.1.1 Risks

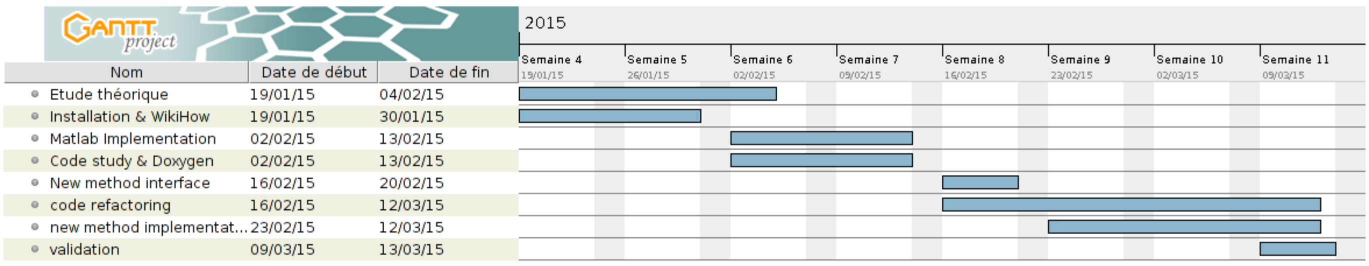
A part of project management and a the project manager's responsibility of is to be able to anticipate the risks of the project. A major issue would be to let the daily events and issues drive the project lifetime. The project manager must be pro-active and produce a risk chart.

As we knew nothing about the topic we were going to work on and we were not really experienced in Fortran and MPI, we originaly anticipate on the following risks.

Definition	**Probability**	**Impact**	**Action**
The product do not fit the customer expectation.	Light	Heavy	Specification with the customer
Resources inadequate	Medium	Heavy	Lighter test creation, request access on computers
Insufficient knowledge	Heavy	Heavy	Increase the time dedicated to each risky task

5.1.2 Final Schedule

Giving the risks anticipated we successfully avoid some issues. However, we didn't anticipated enough on the time spend to debug and to handle MPI library for one part and to refactore the code for a second part. Consequently the initial schedule haven't been precisely observed and we get the following schedule.



Nom	Date de début	Date de fin
Etude théorique	19/01/15	04/02/15
Installation & WikiHow	19/01/15	30/01/15
Matlab Implementation	02/02/15	13/02/15
Code study & Doxygen	02/02/15	13/02/15
New method interface	16/02/15	20/02/15
code refactoring	16/02/15	12/03/15
new method implementat...	23/02/15	12/03/15
validation	09/03/15	13/03/15

Figure 5.1: *Final gantt*

Part II

Reverse Engineering

Chapter 6

Refactoring

6.1 Changing types

The declared types were standardized in order to improve the software quality and portability.

6.1.1 Float numbers

The variables and parameters representing float numbers were standardized. Indeed, some of them were declared as $REAL$ (coded in 4 bytes by default), whereas others were declared as $DOUBLEPRECISION$ or $REAL*8$ (coded in 8 bytes by default). As the memory is not a problem with modern computers, all real numbers are now declared as $DOUBLEPRECISION$.
The changes were made using a simple Java program with String manipulation (*String.replace* method).

6.1.2 *KIND* keyword

As the keyword $KIND$ is interpreted differently depending on the compiler used (for instance, $KIND = 4$ could generate a 4 bytes or 16 bytes variable), all the $(KIND = *)$ were removed. Right now, the real numbers are all coded in 8 bytes, and the integer numbers are all coded in 4 bytes.
The changes were also made using a Java program.

6.1.3 Strings

When using the compiler with the flag *-pedantic*, we discovered that the syntax declaring a string ($CHARACTER*80$ or $CHARACTER*30$) is deprecated. Thus, strings are now declared as following:

$$CHARACTER(LEN = 80)$$

$$CHARACTER(LEN = 30)$$

The changes were also made using a Java program.

6.1.4 Boolean values

Some local variables were declared as $INTEGER$, but only took the values 0 and 1, representing a boolean value (commonly used into a loop). Some were simply removed (the loop has been simplified), and some were replaced by the declaration $LOGICAL$, taking the values $.TRUE.$ and $.FALSE.$.
The changes were made case by case.

6.2 Organizing source code

The source code was cleared in order to improve readability for the developer, and thus improve his ability to maintain the software.

6.2.1 Fortran keywords

The following keywords of Fortran were written in lowercase. So as to have a better visibility of the structure of the code, and to use the same norm as the Fortran developers, they are now in uppercase.
The changes were made using a Java program.

Keywords
PROGRAM, MODULE, CONTAINS, SUBROUTINE, FUNCTION CALL, CONTINUE, RETURN, STOP USE, INCLUDE ALLOCATE, DEALLOCATE OPEN, CLOSE, FILE, UNIT INQUIRE, EXIST, INTRINSIC READ, WRITE, PRINT
IMPLICIT NONE, EXTERNAL TYPE, PARAMETER, INTENT INTEGER, REAL, DOUBLE PRECISION, COMPLEX, CHARACTER, LOGICAL COMMON, POINTER, DIMENSION

GOTO
IF, THEN, ELSIF, ELSE, ENDIF
WHILE, DO, ENDDO, END
SELECT, CASE, DEFAULT

.TRUE., .FALSE. .EQ., .NE. .GT., .LT., .GE., .LE. .OR., .AND., .XOR., .NOT.

6.2.2 Commented code

All the commented algorithmic was removed, to avoid useless lines of code and to ease the readability by the developer.

6.2.3 Semicolons

In Fortran, an end of line indicates an end of instruction. In case of the developer want multiple instructions on one line, he can separate them with the character ';' (semicolon).
But multiple instructions on one line are very difficult to read and understand. It is not a good programming practice. That is why there are no longer multiple instructions per line. Furthermore, all the useless semicolons were removed.

6.2.4 Unused variables

When using the compiler with the flag -*Wextra*, we found some local variables which were unused, probably from older versions of the code. Thus, those variables are not declared any more.

6.2.5 Declarations

The declarations of parameters and variables at the beginning of each subroutine and program were disorganized. Now, there are only one variable or parameter declared per line and the declarations are ordered with the following rules:

- parameters then variables
- for parameters : first "in" parameters, then "in/out" parameters and finally "out" parameters
- order by type: strings, characters, structured types, integers, integer arrays, real numbers, real arrays, then boolean values (subjective order)
- for variables or parameters with the same type, sort by alphabetic order

What is more, declarations are introduced by the following presentation:

```
  IMPLICIT NONE
!############################################
!  DECLARATIONS
!############################################
!#### Parameters ####
!==== IN ====
!=== IN/OUT ===
!==== OUT ====
!#### Variables ####
!############################################
!  INSTRUCTIONS
!############################################
```

6.2.6 Signatures

The order of parameters in the signatures of subroutines was changed to match with the order described in the previous paragraph.
The changes were made using a Java program.

6.3 Renaming and translating

The source code was translated in order to improve readability, and make the code more international.

6.3.1 Comments of the code

All the comments giving further explanations of the algorithmic are now in English for internationalization of the code.

6.3.2 Output messages

All the messages displayed in the console (with the command $PRINT$) are now in English too.

6.3.3 Subroutines, parameters and variables

The subroutines names were translated, standardized (lower_snake_case) and described what the routine does.
The parameters and variables names were also translated and standardized.
The array containing all the old and new names for the routines and parameters can be found in the associated file *refactored_names.ods*.

Chapter 7

Documentation

7.1 Automated comments writing

7.1.1 Issues

The following are the main issues we had to deal with concerning the documentation:

1. **Refactoring dependency**: It is impossible to put Doxygen tags and comments inside the code while we are working on the refactoring.
2. **Repetitive work**: The initial program contained 61 methods and 249 parameters to document. Mistakes in such work are difficult to avoid.
3. **Duplicate parameters** Most of the parameters are similar and it is important to keep coherence in the documentation by having the same description.
4. **Implemented functionalities** The quality of the documentation highly depends on the knowledge of the code we have.

Here are the solutions we had for those problems:

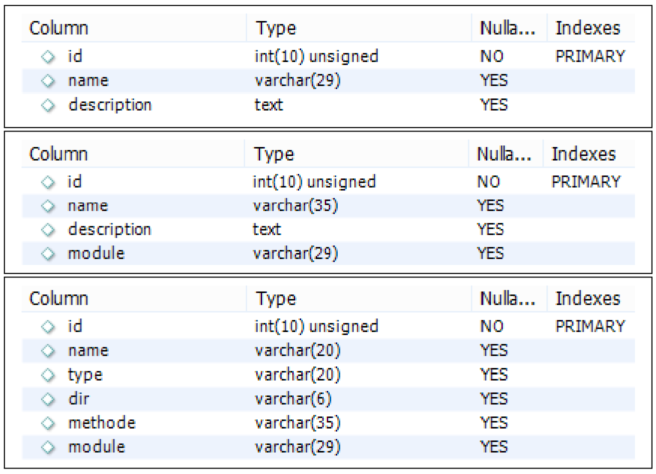
1-2 Automating the comment writing (see subsection 7.1.3)

3 Adding an extra sheet for parameters (see subsection7.1.2)

4 Having a deep look inside the code.

7.1.2 MySQL Database

The MySQL database is basically made to store the information on modules, routines and parameters from the Google Drive. We have 5 tables (see Figure 7.1)



Column	Type	Nulla...	Indexes
id	int(10) unsigned	NO	PRIMARY
name	varchar(29)	YES	
description	text	YES	

Column	Type	Nulla...	Indexes
id	int(10) unsigned	NO	PRIMARY
name	varchar(35)	YES	
description	text	YES	
module	varchar(29)	YES	

Column	Type	Nulla...	Indexes
id	int(10) unsigned	NO	PRIMARY
name	varchar(20)	YES	
type	varchar(20)	YES	
dir	varchar(6)	YES	
methode	varchar(35)	YES	
module	varchar(29)	YES	

id	the key of each table
name	the name of the module, method or parameter
description	the description of the module, method or parameter
module	the name of the module which the method belongs to
type	the type of the parameter
dir	the passing mode of the parameter
method	the name of the method which the parameter belongs to

Figure 7.1: Description of the tables - From top to bottom: **modules**, **methods** and **parameters**

- **modules**: classifies all the modules
- **methods**: classifies all the routines

- **parameters**: classifies all the parameters
- **desc_param**: classifies parameters without duplicate

Each Google sheet is copied locally using Microsoft Excel© and registered into CSV format. Thus, three files are produced: one for modules, one for methods and one for parameters. Then, a simple analysis is performed in order to fill the *desc_param* table with all the parameters without duplicate based on the name and the type. Finally, the result are written back into a dedicated sheet of the Google Drive.

As long as the refactoring evolves, we keept regularly the database up-to-date. Especially for the renaming of the modules, methods and parameters[1]. And because the classifying does not take into account the reordering of the parameters[2], we added the table **type_order** that put a "weight" on each type according to the refactoring.

7.1.3 *AutoComment* software

This small software is the best solution we found to respond to repetitive work issues. This program in Java automatically write headers before each module and method by reading the Fortran files, extracting information from the database and write them back into target files in an other directory. See Algorithm 1 for a brief summary.

Algorithm 1 *AutoComment* algorithm

```
Extract modules from the database
for all module in modules do
    Open the corresponding file
    Create a target file
    Write header for the module in the target file
    for all line in file lines do
        if It is a comment line before a routine then
            Go to the next line
        else if It is a routine declaration then
            Extract information on the routine from database
            Write header for the routine to the target file
        else
            Write the line in the target file
        end if
    end for
end for
```

The program is intend to be launched after the refactoring process. The key idea of this program is that we do not have to write the descriptions twice and we can work on refactoring and documentation at the same time. The algorithm is not fully explained as all the descriptions are not extracted from the database. Methods required advanced description and sometimes references and/or notes. It appeared that it is difficult to format long description in Excel sheets. Because we do not want too long comment lines, we used escape characters but it is not well handled. Besides the use of specific tag for such descriptions (@details, @see, @note) leads to some problem of integration in the database. That is why we choose to put further description of methods into separate text files named as the corresponding methods.

This kind of operations was not useful for the type structures because there are only five of them with few attributes.

7.2 Autogeneration with Doxygen

7.2.1 Doxygen tool parameters

In this subsection, we briefly explain the configuration of the Doxygen documentation generation. In order to make the report clearer, we simply show what differs from the default configuration.

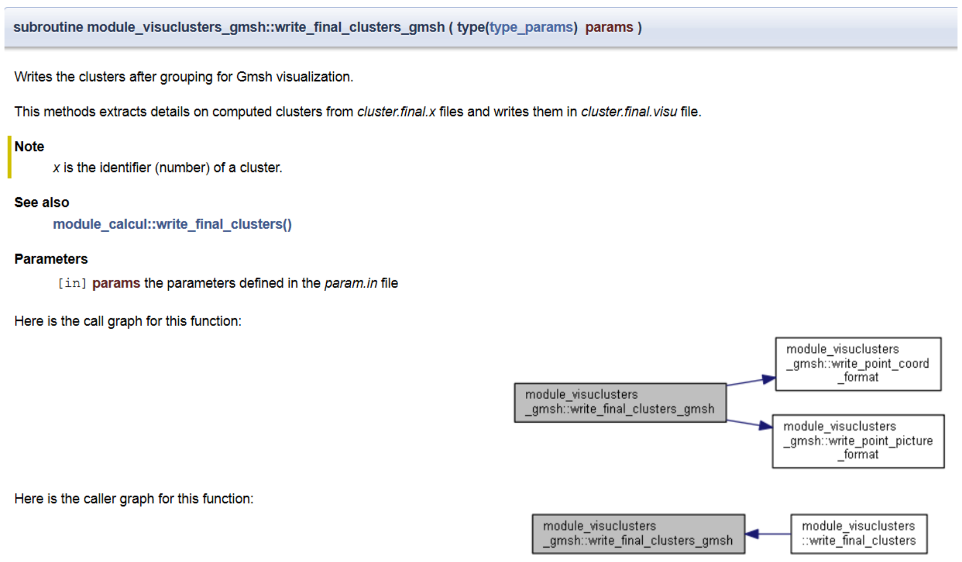
- **PROJECT_NAME**: we set the name of the documentation to "Clustering Methods"
- **PROJECT_NUMBER**: we set the version of the program to "v1.0"
- **FULL_PATH_NAMES**: we disabled this field to have simply the name of the files and not the full path
- **EXTRACT_ALL**: we enabled this field to extract all modules, routines and type

[1]cf section 7.1.3
[2]cf section **??**

- **USE_MATHJAX**: we allowed the use of *MathJax* in order to handle mathematical formula for HTML
- **HAVE_DOT**: we allowed the use of Dot tool from *GraphViz*
- **CALL_GRAPH**: we enabled this field to generate call graphs
- **CALLER_GRAPH**: we enabled this field to generate caller grapgs

7.2.2 Call and caller graphs

The generation of call and caller graphs are handled by Doxygen tool using GraphViz tool. For each routine, it creates the corresponding graphs (see Figure 7.2 for an example). Each called and caller routine in these graphs has an hyperlink to its documentation.



subroutine module_visuclusters_gmsh::write_final_clusters_gmsh (type(type_params) params)

Writes the clusters after grouping for Gmsh visualization.

This methods extracts details on computed clusters from *cluster.final.x* files and writes them in *cluster.final.visu* file.

Note
x is the identifier (number) of a cluster.

See also
module_calcul::write_final_clusters()

Parameters
`[in]` **params** the parameters defined in the *param.in* file

Here is the call graph for this function:

Here is the caller graph for this function:

module_visuclusters_gmsh::write_final_clusters_gmsh
module_visuclusters_gmsh::write_point_coord_format
module_visuclusters_gmsh::write_point_picture_format
module_visuclusters_gmsh::write_final_clusters_gmsh
module_visuclusters::write_final_clusters

Figure 7.2: Example of call and caller graphs

During the documentation generation, an error occurs. It seems that the GraphViz tool does not well create the graphs for the main program.

Part III

Program and Implementation

Chapter 8

Program structure

8.1 Program

As said previously, the code that was provided to us is a parallel code. The master splits the data and send each part to a slave, that computes the clusters and send the result back to the master. Then the master gathers these results.

There are several ways for the slaves to compute the clusters. One of these methods was already implemented in the code: the Spectral clustering. Then we implemented two new methods: the Mean shift and the Kernel k-means. The method is chosen in the parameter file.

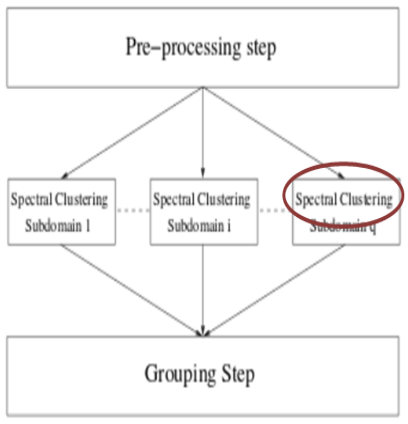


Figure 8.1: Structure of the parallel program

The aim is to split the data points into clusters, the points in the same clusters being as similar as possible. The similarity can depend on the distance for example, or on the color. Here is an example of clustering based on distance.

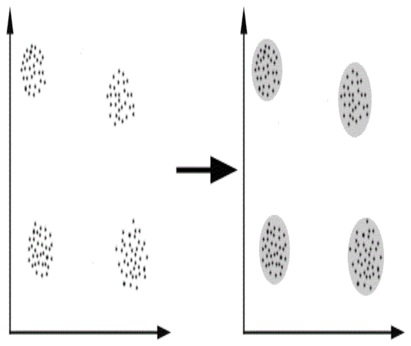


Figure 8.2: Example of clustering based on distance

Chapter 9

Kernel K-Means

9.1 K-Means

The principle of K-means is to select random centers for clusters, split the data set according to these centers and move them until they reach centers of density (see Algorithm 2). This method requires a desired number of clusters.

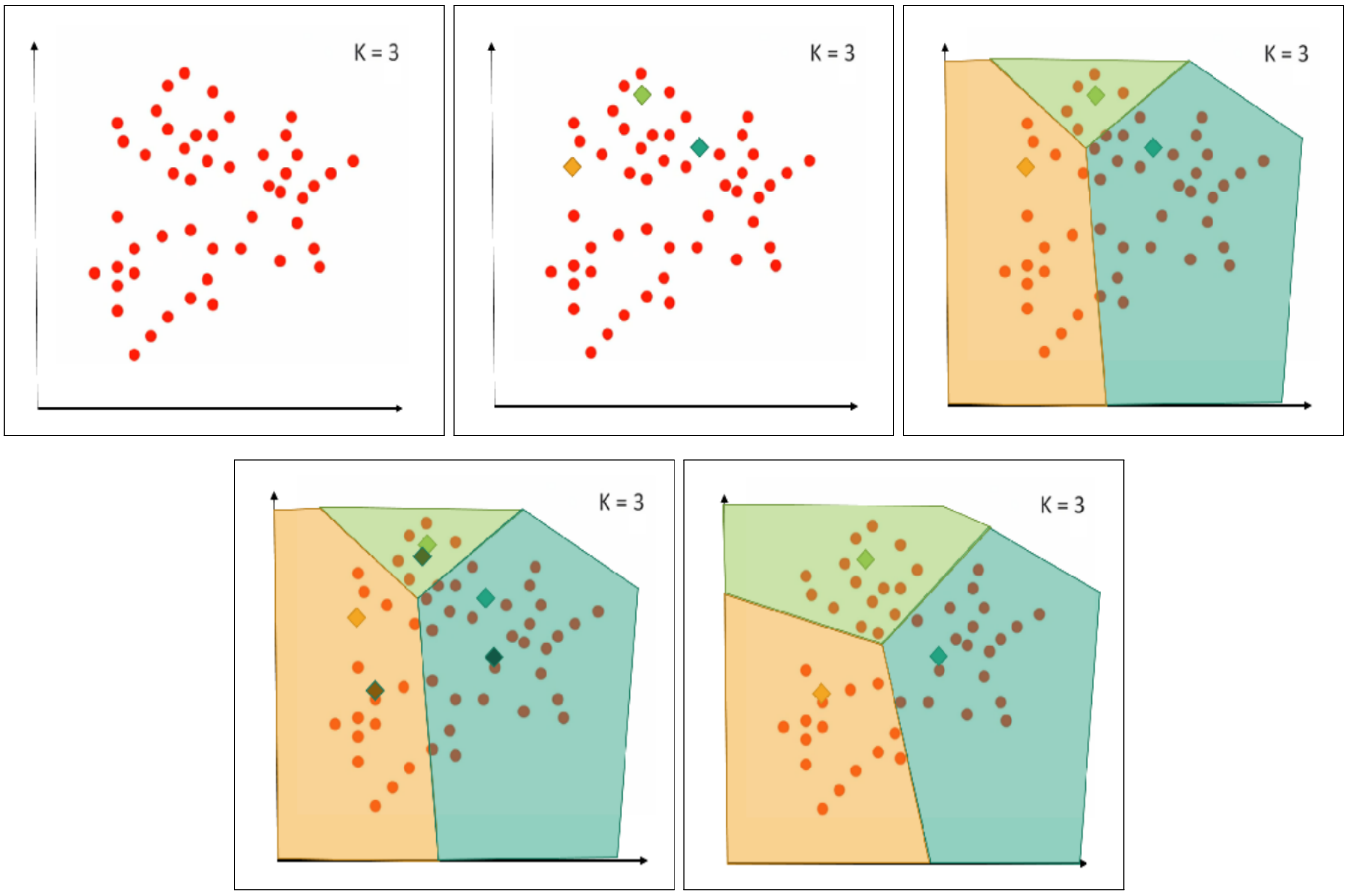


Figure 9.1: K-Mean algorithm - From right to left and top to bottom: Initial data, Select random centers, Split the data, Compute means, Move centers

Algorithm 2 K-Mean algorithm: returns cluster centers

```
loop
    Split the data by matching each point to the closer cluster center
    Compute the mean of each cluster
    if The means and the centers are close then
        return the means
    end if
    Assign mean values to cluster centers
end loop
```

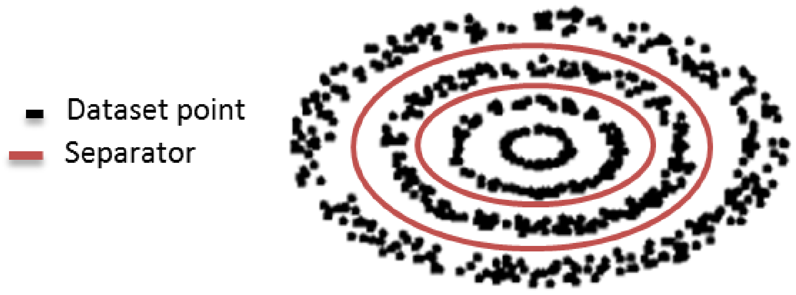


Figure 9.2: Example of non linear separation

However, this method is limited as we can only separated the clusters with hyperplanes, that does not work for non-linear separations (see Figure 9.2). To handle such cases, we pre-process the data by using kernel functions.

9.2 Kernel functions

The main idea of the Kernel K-Mean algorithm is to map the data points into a higher dimensional space in order to lighten non linear separators (see Figure 9.3). For this purpose, we pass the dataset $(x_1, ..., x_n)$ through a Kernel function that computes affinity between point using different approaches from simply finding the shortest distances:

$$\textbf{Polynomial Kernel } K(x_i, x_j) = (x_i^T x_j + \gamma)^\delta \tag{9.1}$$

$$\textbf{Exponential Kernel } K(x_i, x_j) = exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2}) \tag{9.2}$$

$$\textbf{Sigmoïd Kernel } K(x_i, x_j) = tanh(\gamma(x_i^T x_j) + \theta) \tag{9.3}$$

The challenge of the Kernel K-Means algorithm is to select the right Kernel and adjust the parameters γ, δ, σ and θ in order to have the right space. Unfortunately, there is no known computation to find the perfect ones. It exists some methods to find interval that contains the optimal solution for the exponential Kernel but most of the time, the parameters are chosen empirically. That is why we offer the possibility to the user to select value for these parameters.

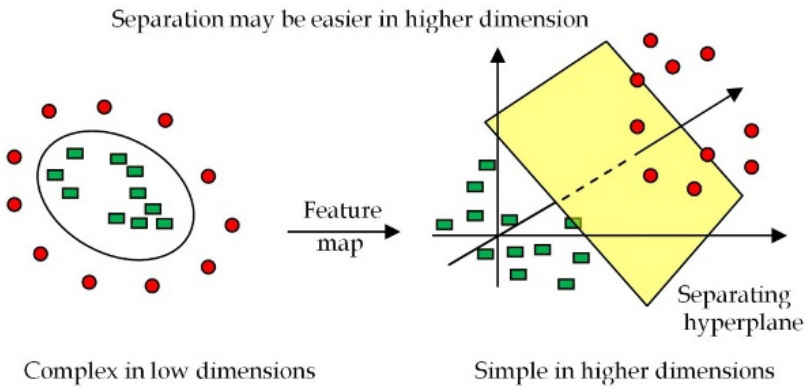

Separation may be easier in higher dimension
Feature map
Separating hyperplane
Complex in low dimensions
Simple in higher dimensions

Figure 9.3: Projection on a higher dimensional space

Chapter 10

Mean Shift

10.1 Main idea

The principle of Mean shift is to move a "window" over the data set following the density gradient (see Figure 10.1). The center of this window has to match a center of density in the dataset. At each iteration, we simply compute the gradient inside the window and then move it along this gradient. We repeat this iteration as long as the gradient norm is significant enough.

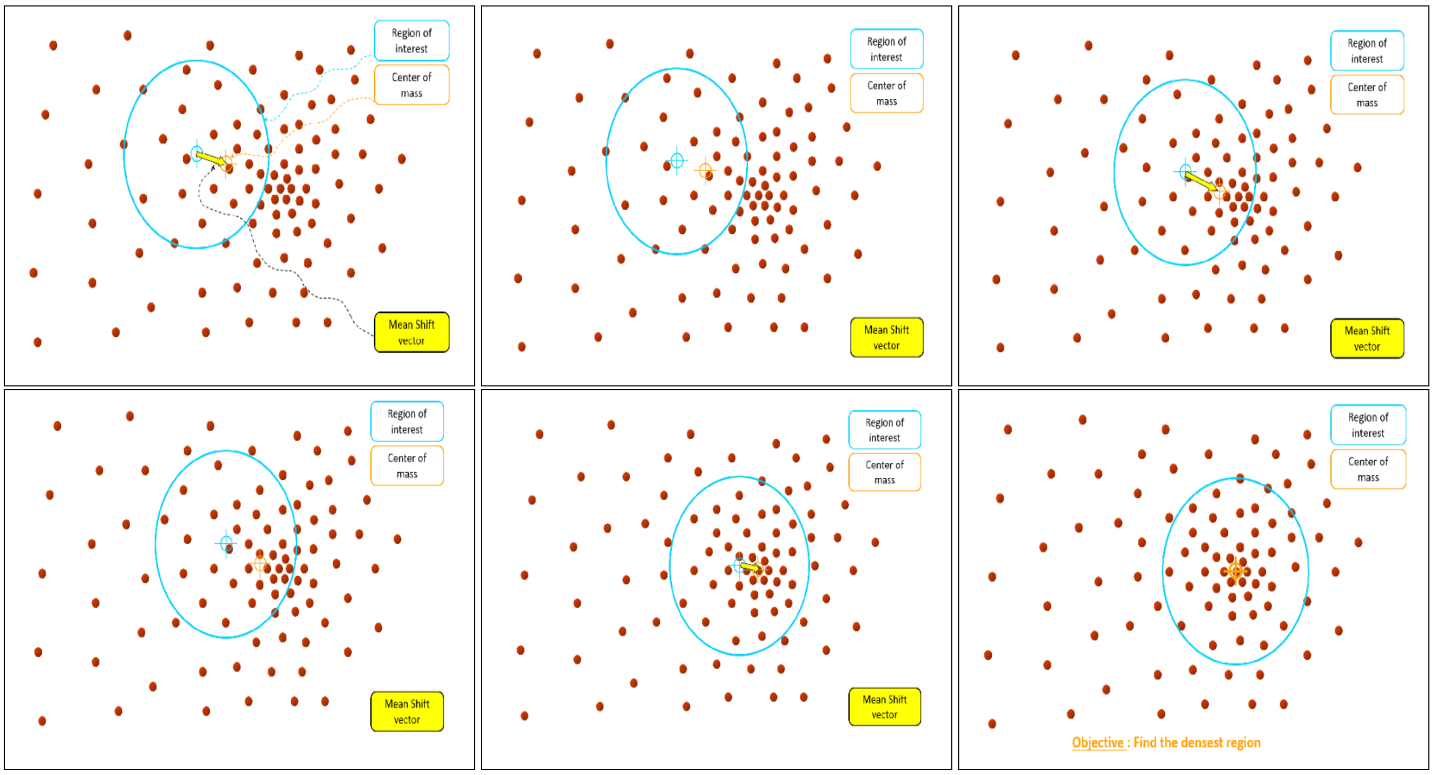


Figure 10.1: Mean shift principle

This simple methodology compute one cluster in the dataset. Then, we apply it again in order to cover all the points. Finally, further computations are made to merge very close clusters.

Chapter 11

Tests and results

11.1 Performances

11.1.1 *3blocsSimple* example

With the following configuration:

- **Number of processes**: 4
- σ: 0.51
- **bandwidth**: 1.5

We get the same results for Spectral clustering and Mean shift(see Figure 11.1). The computation times are also similar:

- **Spectral Clustering**: 0.105 s
- **Mean Shift**: 0.120 s

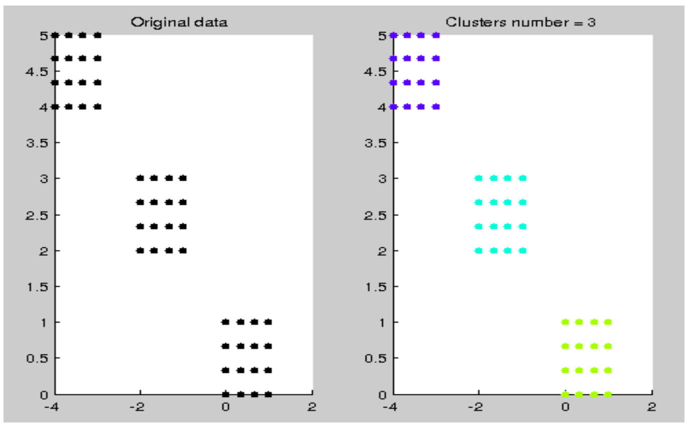


Figure 11.1: *3blocsSimple* result

11.1.2 *croix16* example

With the following configuration:

- **Number of processes**: 1
- σ: 0.51
- **bandwidth**: 1.5

We get the same results for Spectral clustering and Mean shift(see Figure 11.2). However we obtained a faster execution for Mean Shift:

- **Spectral Clustering**: 18.32 s
- **Mean Shift**: 4.106E-2 s

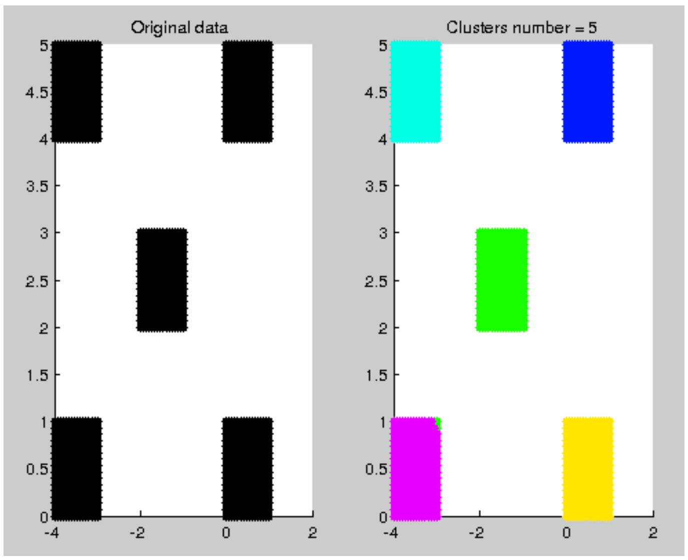


Figure 11.2: *croix16* result

Conclusion

11.2 Conclusion

Although we did not get the expected results, this project successfully achieved its goals. First it taught us really interesting image processing concept while none of us were from the multimedia specialization. Then, it enabled us to get project management concepts which is crucial. Indeed, there are often problem or incomprehension between managers and developers because they do not know each other constraints. Managers have to deal with undecided or unsatisfied client and developers must produce the desired product into a limited time and facing real time modifications some client asks for. As a conclusion we now have a more precise idea of the company-client relationship, what are the traps and the tips to avoid them.