Data clustering optimization with visualization

Fabien Guillaume

MASTER THESIS IN SOFTWARE ENGINEERING

DEPARTMENT OF INFORMATICS

UNIVERSITY OF BERGEN

NORWAY

DEPARTMENT OF COMPUTER ENGINEERING

BERGEN UNIVERSITY COLLEGE

NORWAY

Contents

List of Algorithms:

List of figures:

Chapter 1 - Introduction

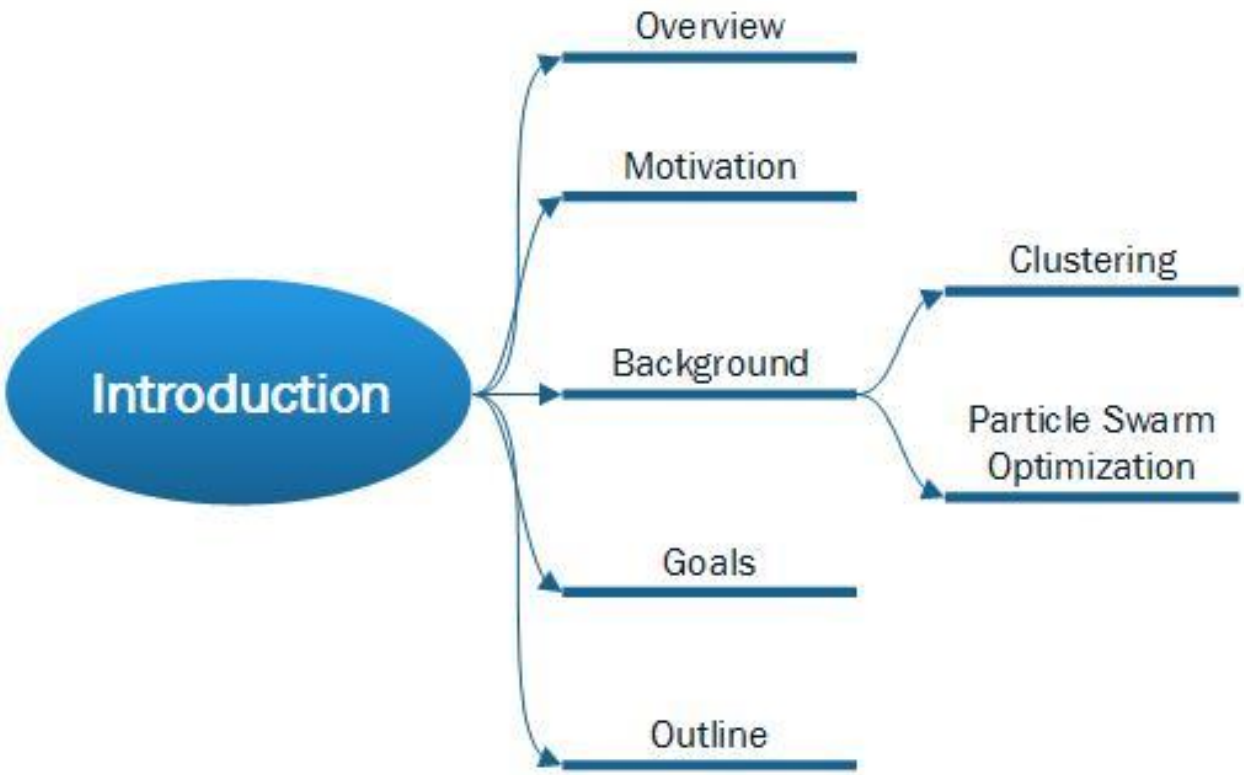


Figure 1.1: Mind map Chapter 1

1.1 Overview

Today the number of sensors (Cameras, microphones, GPS ships, wireless networks…etc.) increases exponentially so does the quantity of data. "Data is the new oil"[1], oil is difficult to extract and refine, likewise, data is hard to extract and refine in order to get value out of it. The problem is that is unstructured and might be huge, for instance mass-spectrometric data may contain thousands of attributes. A manual approach seems to be hopeless, it would cost a lot of highly qualified human resources, time and money. Therefore a new approach is needed to analyze the new raw material that data has become. One possible approach is clustering. This new field of computer science automates the classification of big amounts of data without the need of a priori knowledge. A wide range of research has been done and two new generations of algorithms have emerged, one from the field of statistical analysis that lead to K-means algorithms and the other Kernel based algorithms [1]. The other approach comes from nature's own way of doing things. This field is called Computational intelligence, and implies algorithms like Multi-Layer Perceptron, genetic clustering and so on…

As a result of this new approach the need to optimize the new algorithms generated has arisen The Multi-layer Perceptron has several parameters to be optimized and it is a challenging task to estimate them. Again, optimization is a well-known problem. This problem can be tackled in two different ways. First of all we might consider a very mathematical approach like the simplex method [2]. Similarly to clustering, the other one mimic nature and generates genetic algorithms and Swarm intelligence algorithms. Such an approach allows to efficiently explore a search space to find a near-optimal solution. In this thesis I will focus on Kernel based clustering [1] and Swarm optimization [3].

1.2 Motivation

Clustering analysis is a time consuming and challenging task if done manually. A lot of tools have been developed to automate this task. However, the performance of any clustering algorithms depends largely on the parameters used. Finding the values these parameters is already a difficult problem. The first solution was to use an empirical approach and to adjust manually the parameters. Therefore, the need for a more appropriate solution is needed. Researchers started to use Optimization theory to explore the parameters space of Artificial Neural Network [4]. Another challenge that needs to be addressed is to visualize the optimization algorithm itself. The optimization process takes place in a space of high-dimension. It becomes difficult to visualize the exploration of the search space.

Particle Swarm Optimization (PSO) is a well-known optimization algorithm [3]. It is based on the intrinsic property of swarms to execute complex tasks by the self-organization of simple entities. We speak of intelligence as an emergent property. It is a powerful way to explore a multi-dimensional search space, and find a near optimal solution. In addition, it allows a greater control of the behavior of the algorithm and it enables us to track dynamic optimum.

Kernel Based Clustering is quite a new approach in data-mining [5]. The category of algorithm is based on the following statement: "A complex pattern-classification problem, cast in a high-dimensional

[1] David McCandless: The beauty of data visualization, TEDGlobal 2010

space nonlinearly, is more likely to be linearly separable[2] than in a low-dimensional space provided that the space is not densely populated" [6]. This method of clustering provides excellent results in various problems, with a very efficient time frame.

Optimization techniques have been applied to different data-mining algorithms. However there are few articles about optimized Kernel Based clustering. The lack of such system, the mathematical elegance of Kernel based clustering in addition to simple efficiency of Particle Swarm Optimization (PSO) has triggered our interest in designing such system.

1.3 Background

1.3.1 Clustering

Data clustering is often confused with classification. However, in classification the classes in which objects need to be assigned are predetermined. In clustering, these classes have to be defined too. As mentioned earlier, the clustering problem has been studied for many years. Even so, there is no uniform definition of it. Generally, by data clustering, we mean that for a given data-set and a given similarity measure, we regroup data points such that object in the same group are similar, and objects in different groups are dissimilar, according to the similarity measure.

Clustering is applied to many problems, such as:

- Gene expression data [7].
- Image segmentation [8].
- Market research [9].
- And many others…

The creation of groups is based on the concept of similarity and distance. These are reciprocal concepts. Similarity coefficients describes quantitatively how similar two data points are. The greater the similarity coefficients, the greater the similarity. Distance measure, on the other hand, describes quantitatively how different two data points are. The greater the distance, the greater the dissimilarity. We will define more precisely these notions in the next chapter. To summarize, a cluster is a set of objects or data points that share the same properties i.e. that have a small mutual distance.

Every clustering algorithms share a common structure:

[2] Linearly separable: a hyper plane can discriminate the data points of two or more classes.

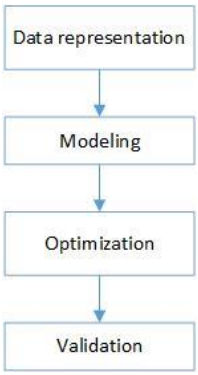


Figure 1.2: Process of data clustering

The data representation determines the structure of the clusters to be discovered in the data. Followed by the modeling phase that defines the notions of cluster and the separating criteria between clusters. The optimization phase refines the clusters and finally the result is validated.

1.3.2 Particle Swarm Optimization

Biologist have been studying the unpredictable choreography of a bird flock and especially the ability to fly synchronously and recover their formation after a change of direction. They tried to simulate graphically this phenomena. From this initial objective, the concept evolved into a population-based search algorithm, the particle swarm optimization algorithm [3].

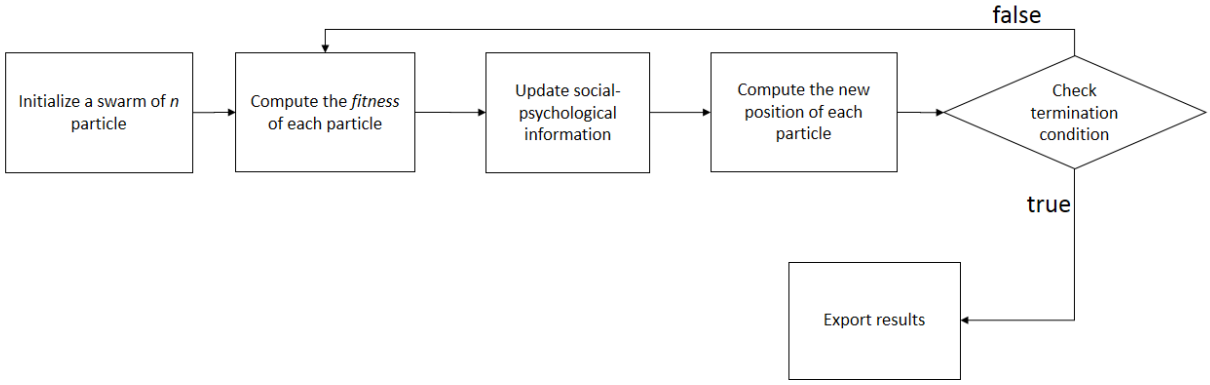


Figure 1.3: Basic Structure of PSO

In PSO, birds are assimilated to particles and are flown through hyper-dimensional search space. The social-psychological tendency of individuals is used to update the velocity of particles. In other words, each particles is influenced by the experience of others particles, its neighbors, and by its own knowledge. This means that the behavior of the swarm is determined by each particle itself. This property is called symbiotic cooperation. The modeling of this behavior encourages the particles to go back to successful region of the search space. The basic structure of a PSO algorithm is presented in

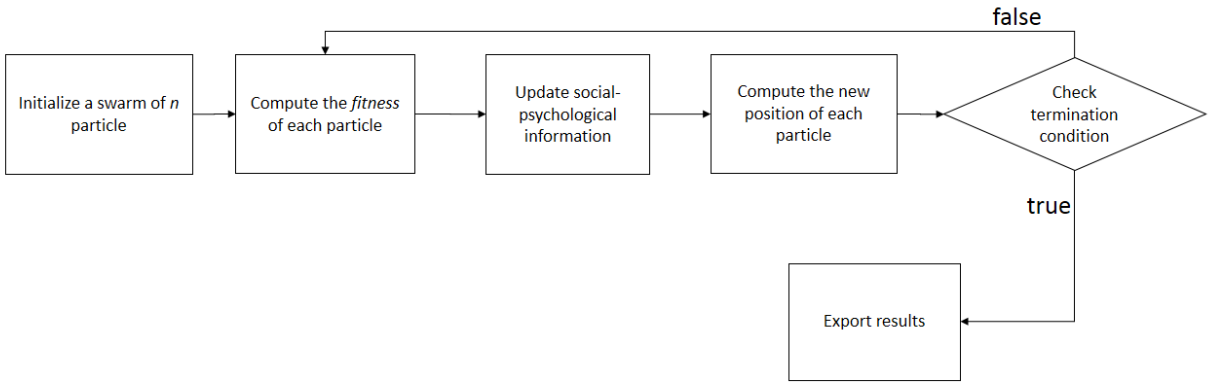


Figure 1.3.

false
Initialize a swarm of *n* particle
Compute the *fitness* of each particle
Update social-psychological information
Compute the new position of each particle
Check termination condition
true
Export results

Figure 1.3 Two elements are critical when designing a PSO. The first is the fitness function which is highly problem dependent, and is a measure of how well a particle performs. The second is the update of the position, it is determined by the velocity, and in this case, there are various approaches. But it is important to remember that the way how the velocity is computed strictly determines the behavior of the swarm. Figure 1.3 just shows the very basic idea behind PSO, there are numerous variations of this structure. The choice of a specific variation depends of the problem to be solved.

1.4 Goals

This thesis aims to exhibit the potential of swarm optimization and its ability to work on different tasks. We will applied particle swarm optimizer to kernel based clustering and different study cases. From these experiments we will draw conclusions about PSO's abilities. We want to study the potential and limits of such optimization approach.

Also, to demonstrate the flexibility of such an algorithm, Swarm optimization will be applied to a model of dynamic system. This will prove the ability of Swarm Optimization to work in complex and highly non-linear search space. Finally a critical objective is to visualize the experiments. The system will have a visualization tool to represent the optimization process and a Graphical User Interface.

1.5 Outline

Chapter 1: Introduction: This chapter simply describes the overall idea, introduces a few definitions and the outline of the thesis.

Chapter 2: Kernel clustering: This chapter presents the theoretical knowledge around Kernel based algorithm and Kernel clustering.

Chapter 3: Particle Swarm Optimization: The theory behind Particle Swarm Optimization is detailed in this chapter.

Chapter 4: Design of the system: The requirements and the design of the system are presented.

Chapter 5: User interface: The user interface of the system is exhibited and explained in this chapter.

Chapter 6: Study case – the movie model: This chapter presents a dynamic model, called the movie model. We show the need of an optimization algorithm to improve such a model, and perform experiments.

Chapter 7: Study case – DNA classification: We presents the problem of DNA classification using Artificial Neural Network and Support Vector Machine, and apply Particle Swarm Optimization to find the appropriate parameters.

Chapter 8: Study case – Support Vector Clustering: Particle Swarm Optimization is applied to Support Vector Clustering. The data set used is presented and the different implications of such experiment are presented in this chapter.

Chapter 9: Conclusion and further work: A summary of the experiments is done. We present our conclusion and the evolution of this project in the future.

Chapter 2 - Kernel clustering

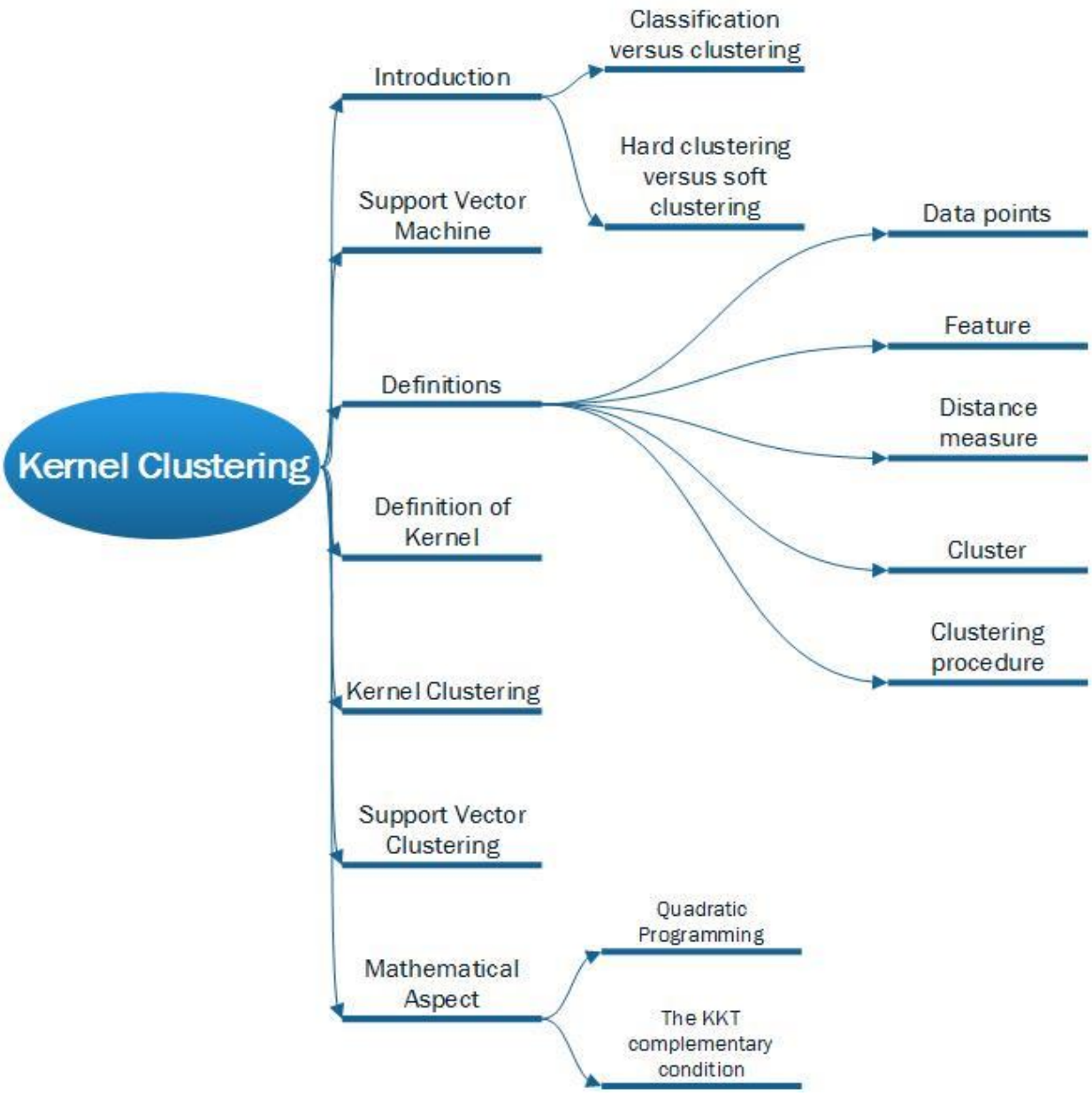


Figure 2.1: Mind map Chapter 2

2.1 Introduction

2.1.1 Classification versus clustering

Clustering and classification are based on the same concept similarity grouping (i.e. share common properties) according to specific criteria. Actually, this is one of the oldest human ability, and can probably be extended to more species [11]. In order to learn something, people try to find descriptive features of an unknown object and compare them to known object according to their similarities and/or dissimilarities. This allows people to classify information, for instance natural objects are splitted in three groups, animals, plants, minerals. Each of these groups is further divided into subgroups and so on…

Data analysis naturally converged towards clustering and classification to provide a basis for further reasoning, decisions making and understanding any kind of phenomena. Let us consider a clinical treatment, a specific disease might have different subtype sharing similar symptoms but responding differently to the same cure. Cluster analysis of gene expression data measuring the activity of genes can be a solution to discovering subtypes and adapt the therapy [7] [12].

Cluster analysis is divided into two main domains: *Classification* and *Clustering*. In Classification data is categorized in groups known a priori while the clustering process has to discover the groups in the data. In other words clustering needs to find the common features between data by itself. The latter method is a tool of choice to explore unknown data and extract knowledge from it.

2.1.2 Hard clustering versus soft clustering

As expected there are two types of clustering: *hard* and *soft*.

Given a set of input data $X = \{x_1, \dots, x_j, \dots, x_N\}, where\ x_j = \{x_{j1}, x_{j2}, \dots, x_{jd}\} \in R^d$ with each x_{ji} is called a property, or features, *d* is the dimensionality of *X* and *N* is the cardinality of *X*.

A hard clustering process will seek to create k partition of X, $C = \{C_1, \dots, C_k\}$ with $k < N$ subject to the following constraint:

- $\mathrm{C_i} \neq \emptyset, \mathrm{i} = 1 \dots \mathrm{k}$
- $\bigcup_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{C_i} = \mathrm{X}$ *Equation 2.1*
- $\mathrm{C_i} \cap \mathrm{C_j} = \emptyset,\ \mathrm{i,j} = 1 \dots \mathrm{k}, \mathrm{i} \neq \mathrm{j}$

It means that in hard clustering each data points is assigned to one and only one cluster.

Soft clustering on the other hand allows input patterns to belong to all k clusters with a degree of membership $u_{ij} \in [0, 1]$, u_{ij} represents the membership degree of object j to cluster i. The membership function is subject to two constraints:

- $\sum_{i=1}^{k} u_{ij} = 1, \forall j$
- $\sum_{j=1}^{N} u_{ij} <$ *Equation 2.2*
  $N, \forall i$, N is the number of data points

Soft clustering is also called fuzzy clustering [13]. The next section will clarify the vocabulary and give definition of it.

2.2 Definitions

2.2.1 Data points

A data point or object, also called input pattern represents a coherent group of informations, itself composed of single units of information called features or properties. In order to be processed, data points are converted in vectors where the dimension is the number of features.

For example, in the case of face recognition, the vector would be a single picture of a face then the features would be the pixels on that image. The dimension of each input pattern becomes very high, very quickly depending of image's resolution.

Data points are noted as follow:

$x_j = (x_{j1}, x_{j2}, \dots, x_{jd})^T$, where d is the number of features (i.e. the dimension).

A data set is simply a collection of input patterns and its notation is:

$$X = \{x_1, \dots, x_j, \dots, x_N\}$$

For N data object with d features, an N*d pattern matrix is built from the corresponding vectors.

2.2.2 Features

Features are classified in three categories: *continuous*, *discrete* and *binary*. Continuous features take values in an infinite range of sets, such as the weight of humans, the concentration of a molecule in a blood sample, the frequency of a specific codon in a specific gene, etc… Discrete features have a countable infinite number of values, such as the color of the eyes, the country of a person, the brand of a car. Binary features are a special case of discrete properties, they can take only two values and are also called dichotomous features. It can be a Boolean value for instance.

2.2.3 Distance measure

When an algorithm is clustering the pattern matrix, it has to deal with measuring the similarity or dissimilarity between two patterns. However, with real world data, some or all the input patterns can have missing values. If the number of incomplete patterns is small compared to the size of the input set, then they can be discarded. Very often this is not the case and the major part of a set can have missing features. Therefore the solution is to consider the contribution of the missing features to the proximity measure as null. The distance between object x_i and x_j is defined by:

$$D(x_i, x_j) = \frac{d}{d - \sum_{l=1}^{d} \delta_{ijl}} * \sum_{all\ l\ and\ \delta_{l=0}} d_l(x_{il}, x_{jl}) \qquad \textit{Equation 2.3}$$

Where $d_l(x_{il}, x_{jl})$ is the distance of feature l between object i and j and

$$\delta_{ijl} = \begin{cases} 1, if\ x_{il}\ or\ x_{jl}\ is\ missing \\ 0, \quad otherwise \end{cases} \qquad \textit{Equation 2.4}$$

A distance or dissimilarity function on a data set X must satisfy the following properties:

- Symmetry:

$$D(x_i, x_j) = D(x_j, x_i) \qquad \textit{Equation 2.5}$$

- Positivity:

$$D(x_i, x_j) \geq 0, \forall x_i, x_j \qquad \textit{Equation 2.6}$$

- Triangle inequality:

$$D(x_i, x_j) \leq D(x_i, x_k) + D(x_k, x_j), \forall x_i, x_j, x_k \qquad \textit{Equation 2.7}$$

- Reflexivity:

$$D(x_i, x_j) = 0\ iff\ x_i = x_j \qquad \textit{Equation 2.8}$$

In this thesis, we will work with continuous variables and the distance measure will be the Euclidian distance. This proximity measure is a special case of the Minkowski distance, called L_p that is defined by:

$$L_p = D(x_i, x_j) = \left(\sum_{l=1}^{d} \left(|x_{il} - x_{jl}|^p \right) \right)^{1/p} \qquad \textit{Equation 2.9}$$

The Euclidian distance is L_2 is a special case when p=2, given by:

$$L_2 = D(x_i, x_j) = \left(\sum_{l=1}^{d} \left(|x_{il} - x_{jl}|^2 \right) \right)^{1/2} \qquad \textit{Equation 2.10}$$

There are two other well-known cases of the Minkowski distance, the Manhattan distance or $L_1$and the sup distance or L_∞.

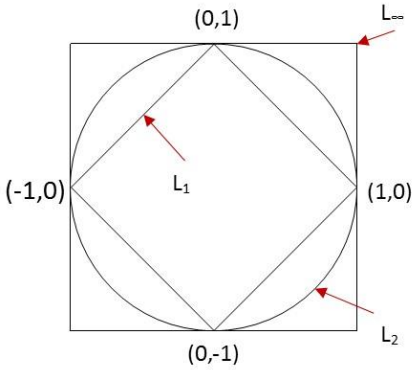


Figure 2.2: Illustration of the Minkowski distance

L_2 satisfies the conditions given in Equation 2.5 to Equation 2.8. However, features can be measured in various units and the variance of certain properties or their large value can bias their weight in a data set [14]. To solve this problem Hogg and Tanis [15] proposed a method called *data standardization*, where each feature has zero mean and unit variance:

$$x_{il} = \frac{x_{il}^* - m_l}{s_l}, i = 1 \dots N, l = 1 \dots d \qquad \textit{Equation 2.11}$$

x_{il} is the raw data, m_l *is the sample mean*:

$$m_l = \frac{1}{N}\sum_{i=1}^{N} x_{il}^* \qquad \textit{Equation 2.12}$$

While s_l is the sample standard deviation:

$$s_l = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_{il}^* - m_l\right)^2} \qquad \textit{Equation 2.13}$$

Another normalization exists, based on the extrema of the data. It puts all features in the interval [0, 1]:

$$x_{il} = \frac{x_{il}^* - min(x_{il}^*)}{max\left(x_{il}^*\right) - min\left(x_{il}^*\right)} \qquad \textit{Equation 2.14}$$

With the two last definitions at hands, we may try to define the concept of a cluster.

2.2.4 Cluster

The goal of a clustering algorithm is to partition a data set X into a certain number of clusters, i.e. subsets of X where each member of a given subset shares common properties (have a small distance between

each other). However, there is no universally admitted definition of the term cluster. According to Britts & all [11] a “formal definition is not only difficult, but may even be misplaced”.

According to the dictionary a cluster is “a group or bunch of several discrete items that are close to each other”.

In [16] Everitt states three other definitions given by:

1. A cluster is a densely populated region of the feature space separated by relatively low density region of this space.
2. “A cluster is a set of entities which are alike, and entities from different clusters are not alike.”
3. “A cluster is an aggregate of points in the test space such that the distance between two points in the cluster is less than the distance between any point in the cluster and any point not in it.”

In other words, a cluster is a subset of the input data points that are similar. This similarity between data points is obtained using a distance measure. The data point from another subset are not similar, that means that the distance between two points from different cluster is larger than the maximum inner distance of each cluster.

The last aspect that might be worth considering is the granularity of the clustering.

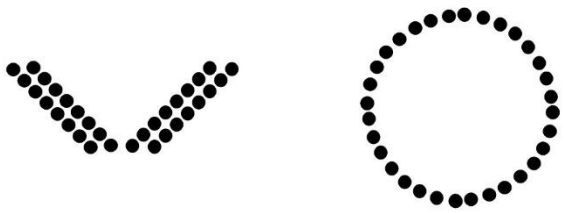


Figure 2.3: Clustering granularity

In Figure 2.3 one might see two clusters or three, both answers are right.

2.2.5 Clustering procedure

It is commonly admitted that the clustering procedure is divided into four major steps [12] as shown in Figure 2.4.

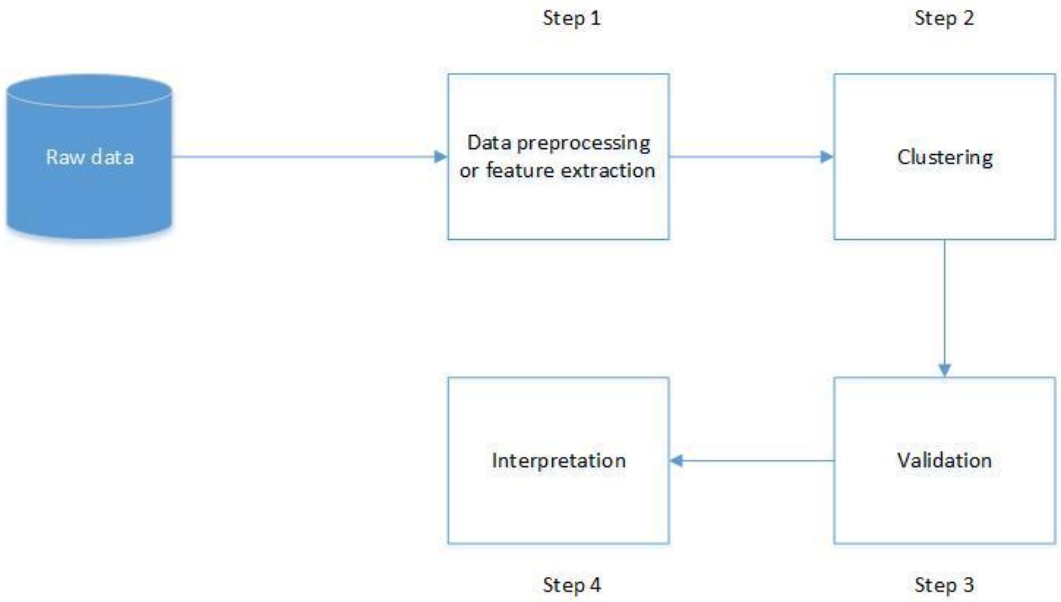


Figure 2.4: Clustering procedure

- Step 1: Feature extraction applies transformations to extract or generate new features from the original ones. This method could be more efficient for uncovering the data structure.
- Step 2: Clustering algorithm design determines the proximity measure.
- Step 3: Validation, Clustering algorithms can create clusters whether they exist or not in the data. These steps are an attempt to evaluate the algorithm's performance.
- Step 4: Interpretation of the results, experts in the relevant field try to interpret the results and to see if the clustering provided meaningful information on the original dataset, or not.

In the next section we briefly introduce the Support Vector Machine (SVM) to lay the foundation for the Support Vector Clustering (SVC). In the description of the SVM we want to avoid to use too much mathematics, the goal is to give an overview of the SVM. The mathematical formulation will come in the description of the SVC.

2.3 Support Vector Machine

The Support Vector Machine [17] is a binary learning machine, i.e. it can classify two classes of pattern. In the context of pattern classification, the core idea can be summarized as follows:

Given a training sample, SVM constructs a hyper plane (decision surface) to discriminate between positive and negative samples in order to maximize the margin of separation, Figure 2.5 illustrates this concept.

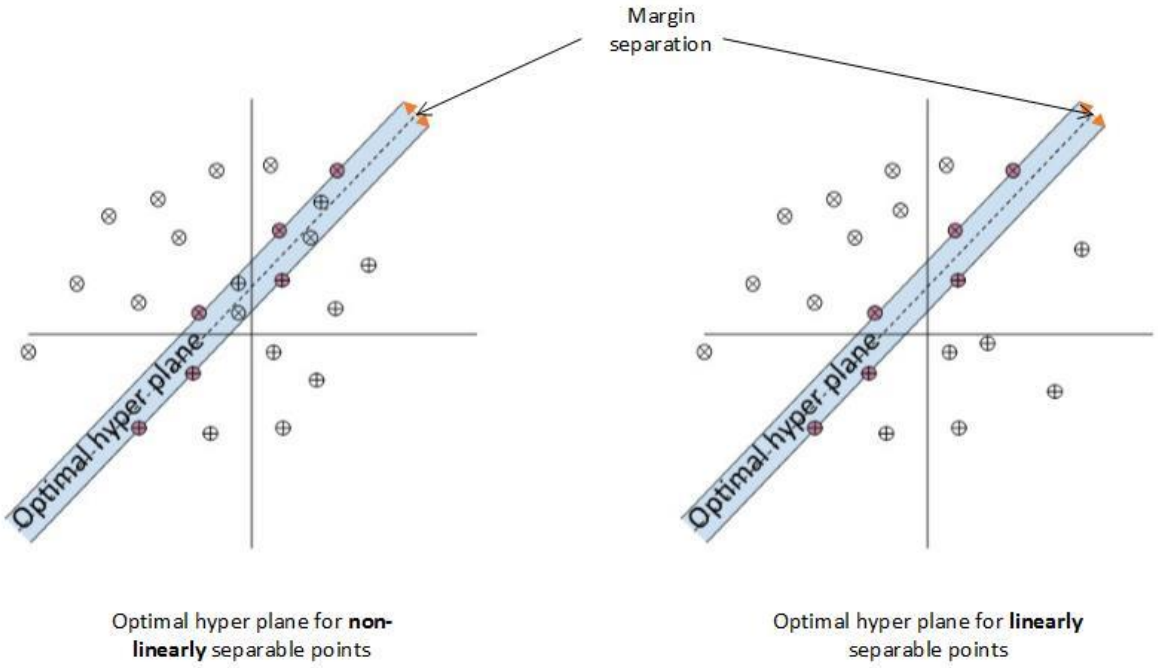


Figure 2.5: Margin separation

Support Vectors are a subset of data points selected by the learning algorithm. The machine is based on the *inner product kernel* between a support vector and a vector drawn from the training data. Because of this property, SVM is referred to as a *kernel method*. This kernel method designs an optimal decision surface. The optimality is rooted in convex optimization [18]. However, such a feature comes at the price of a high computational complexity. The *kernel trick* [19] solves the problem of computational complexity. The *kernel trick* is detailed in section 2.6.

SVM projects training points into a feature space with a much higher dimension than the input space. Such projection is non-linear, but according to the Cover Theorem [6], the data may become linearly separable in the feature space. This means that the data points can be separated by the hyper plane in the feature space. The non-linear map is unknown and is computationally expensive, and sometimes become virtually impossible to compute. The kernel trick is then used to compute the images of the training points in the feature space. This trick avoids the computation of the non-linear map. Let a nonlinear map $\Phi: R^D \rightarrow F$ where F represent a feature space and k be a Mercer's Kernel [20], we can replace the inner product of Φ defined by:

$$\Phi: R^D \rightarrow F \qquad \textit{Equation 2.15}$$

$$k(x, x') = \Phi^T(x')\Phi(x) \qquad \textit{Equation 2.16}$$

Mercer's kernel are known mathematical functions (polynomial, sigmoid etc…), therefore we can calculate the inner product of Φ without knowing it. In the feature space the learning algorithm selects the support vector to build the decision surface and map it back in the input space. The second mapping is achieved by first solving a convex optimization problem then applying a linear mapping from the feature space to the output space. Figure 2.6 illustrates the concept of SVM.

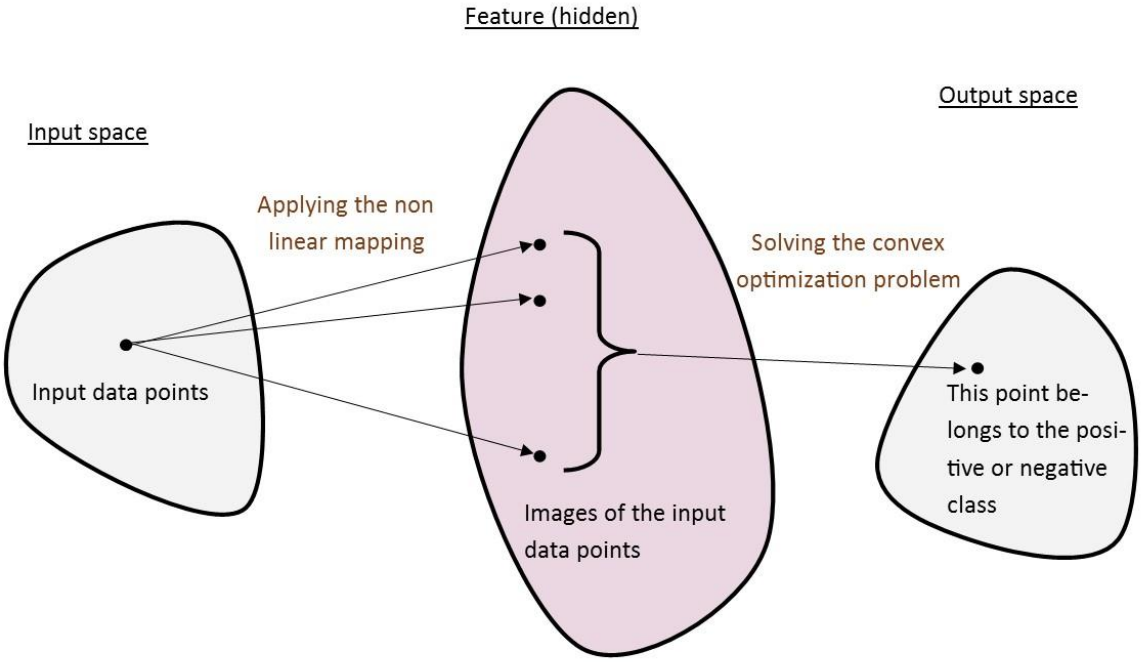


Figure 2.6: Illustration of the two mappings in the SVM.

SVM has been initially designed to classify binary data. Later the Multi Class SVM was created allowing the classification of a finite number of classes. But supervised learning assumes a priori knowledge on the data (the number of classes). Hence, the Support Vector Clustering [5] was developed which is an unsupervised learning algorithm using the core idea of SVM. We will explore the SVM with kernel clustering in details in the next sections.

2.4 Kernel Clustering

The introduction of the Support Vector Machine [1] has increased the popularity of the Kernel algorithm since 1990s, due to its high performance in both supervised classification and regression analysis. It has been successfully applied in unsupervised classification or clustering since then. The key behind that is the Cover Theorem [6]:

> *A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.*

Given a set $X = \{x_1, \dots, x_j, \dots, x_N\}$ $with\ x_j \in R^d$ of input pattern and a nonlinear map $\Phi: R^D \to F$ where F represents a feature space of arbitrarily high dimensionality. X is mapped into F through Φ so that a linear algorithm can be performed. The Cover Theorem is illustrated in Figure 2.7.

A problem soon appears, as the dimensionality increases, so does the complexity. It is called the curse of dimensionality [21]. This difficulty is overcome using the kernel trick from Mercer's theorem [20]:

Let $k(x, x')$ be a continuous symmetric kernel that is defined in the closed interval $a \leq x \leq b$, and likewise for x'. The kernel $k(x, x')$ can be expanded in the series:

$$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \varphi_i(x) \varphi_i(x') \qquad \textit{Equation 2.17}$$

With positive coefficients $\lambda_i > 0$ for all i. For this expansion to be valid and for it to converge, it is necessary and sufficient that the condition:

$$\int_b^a \int_b^a k(x, x') \psi(x) \psi(x') dx dx' \geq 0 \qquad \textit{Equation 2.18}$$

holds for all $\psi(.)$, for which we have:

$$\int_b^a \psi^2(x) dx < \infty \qquad \textit{Equation 2.19}$$

where a and b are the constants of integration

Calculating Φ is time-consuming and often infeasible. However, Mercer Theorem allows us to avoid this computation and there is no need to explicitly describe the nonlinear mapping Φ neither the image points in the feature space F. This technique is known as the Kernel trick.

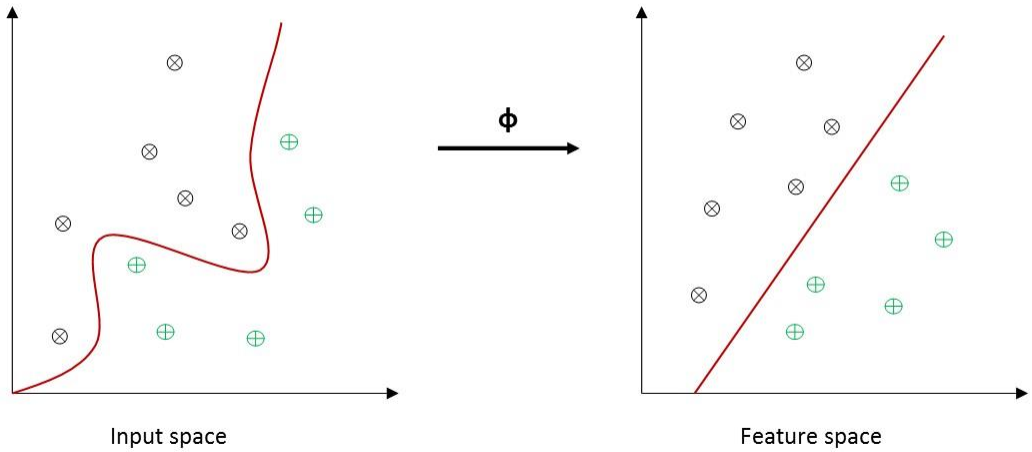


Figure 2.7: The Cover Theorem

In the next section, the notion of Kernel is explained with different types of kernels.

2.5 Definition of Kernel

The function $k(x, x')$ is called an inner product kernel, or kernel.

> *It is a function that computes the inner product of images produced in the feature space under Φ of two data points in the input space [22].*

Two properties are derived from this definition:

Property 1: $k(x, x')$ is symmetric about the center x':

$$k(x, x') = k(x', x) \qquad \textit{Equation 2.20}$$

The maximum is reach for $x = x'$.

Property 2: The volume under $k(x, x')$ $is\ a\ constant$.

However, this is the general definition of a kernel. To use the kernel trick, the function $k(x, x')$ has to satisfy Mercer's theorem. In the scope of this observation, only four types of kernel remains, they are listed below.

The polynomial kernel:

$$k(x, x') = (x * x' + 1)^p \qquad \textit{Equation 2.21}$$

The Gaussian kernel:

$$k(x, x') = exp\left(-\frac{1}{2\sigma^2}\|x - x'\|^2\right) \qquad \textit{Equation 2.22}$$

where σ is the width parameter or margin separation. In the case of clustering it controls the granularity of the clustering.

The sigmoid kernel:

$$k(x, x') = \tanh(\theta_0(x + x') + \theta_1) \qquad \textit{Equation 2.23}$$

where θ_0 and θ_1 are user-specified parameter. Furthermore, the sigmoid kernel satisfies Mercer's theorem only for specific value of θ_0 and θ_1.

In the case of Support Vector Clustering, the Gaussian kernel is the most used because the polynomial kernel does not allow tight contours of the clusters [23].

With all the previous definitions, we can now present the Support Vector Clustering algorithm (SVC).

2.6 Support Vector Clustering

Over the last years various Kernel clustering algorithms have been developed. They may be grouped in three families. The first type is based on *Kernelising the Metric*, the metric is computed by means of Mercer's kernel in the feature space. The second implements K-means in the feature space using *Kernel Methods*. The last one is based on Support Vector Machine including *Kernel Methods*. In this section we introduce the Support Vector Clustering algorithm and explain the Kernel trick.

In SVC, the data points are mapped from data space into a high dimensional feature space by means of a Mercer's kernel (here the Gaussian kernel). Then we look for the smallest hyper-sphere capturing all the image data points. Once mapped back into the data input space, the contour of the sphere describes the boundaries of clusters. Finally, data points enclosed by the same closed contours are assigned to a cluster. This is done by computing the adjacency matrix[3].

Given a set $X = \{x_1, \dots, x_j, \dots, x_N\}$ $where\ x_j \in R^d$ of input pattern and a nonlinear map $\Phi: R^D \to F$. The object is to find the smallest hyper-sphere in F capturing all the data points of X after the nonlinear mapping. In other words, we need to find the hyper sphere H with the minimal radius R such as:

$$\left\|\Phi(x_j) - \alpha\right\|^2 \leq R^2, \forall j \qquad \textit{Equation 2.24}$$

Where $\|.\|$ is the Euclidian distance, α is the center of H. The introduction of the slack variables ξ_j incorporates soft constraints. Slack variables replace inequality constraints by equality constraints plus non-negativity constraints. Equation 2.24 becomes:

$$\left\|\Phi(x_j) - \alpha\right\|^2 \leq R^2 + \xi_j \text{ , where } \xi_j \geq 0 \qquad \textit{Equation 2.25}$$

With $\xi_j \geq 0$. The primal problem is solved in its dual form by introducing the Lagrangian.

$$L = R^2 - \sum_j \left(R^2 + \xi_j - \left\|\Phi(x_j) - \alpha\right\|^2\right)\beta_j - \sum_j \xi_j \mu_j + C\sum_j \xi_j \qquad \textit{Equation 2.26}$$

With $\beta_j \geq 0$ and $\mu_j \geq 0$ are Lagrange multipliers and $C\sum_j \xi_j$ is a penalty term and C a constant. Applying the stationary condition to L, i.e., setting to zero the derivative of L with respect to R, α and ξ_j leads to:

$$\sum_j \beta_j = 1 \qquad \textit{Equation 2.27}$$

$$a = \sum_j \beta_j \Phi(x_j) \qquad \textit{Equation 2.28}$$

[3] An adjacency matrix is a matrix representing the connection of nodes in a graph. It is just one of the many representations of a graph.

$$\beta_j = C - \mu_j \qquad \textit{Equation 2.29}$$

This is a quadratic optimization problem [18] which involves the Karush-Kuhn-Tucker complementarity conditions (see section 2.7.2) [24] we have:

$$\xi_j \mu_j = 0 \qquad \textit{Equation 2.30}$$

$$\left(R^2 + \xi_j - \left\|\Phi(x_j) - \alpha\right\|^2\right)\beta_j = 0 \qquad \textit{Equation 2.31}$$

From Equation 2.31 we observe that the image of a point x_j with $\beta_j > 0$ and $\xi_j > 0$ is outside the hyper sphere H. According to Equation 2.30, $\mu_j = 0$. It follows now from Equation 2.29 that $\beta_j = C$. Such a point is called a *Bounded Support Vector* (BCV). A point x_j with $0 < \beta_j < C$ then from Equation 2.29 $\mu_j > 0 \rightarrow \xi_j = 0$. Such a point lies on the feature space sphere and is called Support Vector (SV). All other points are inside the sphere.

SVs lie on cluster boundaries while BSVs are outside, the rest are in the clusters. Using these relations we can remove R, α and μ_j and rewrite Equation 2.26 into its dual expression:

$$W = \sum_j \Phi(x_j)^2 \beta_j - \sum_{ij} \beta_i \beta_j \Phi(x_i).\Phi(x_j) \qquad \textit{Equation 2.32}$$

The variables μ_j are replaced by the following constraints:

$$0 < \beta_j < C, j = 1, \dots, N$$

At this stage we still don't know the nonlinear map nor the Lagrange Multipliers. However, according to Mercer's theorem the dot product $\Phi(x_i).\Phi(x_j)$ may be replaced by a Mercer kernel $K(x_i, x_j)$ as

$$K(x_i, x_j) = \Phi(x_i).\Phi(x_j) \qquad \textit{Equation 2.33}$$

This replacement is known as *the Kernel trick*. The dual Lagrangian [25] W is now written as:

$$Max_{\beta_j} W = \sum_j K(x_j, x_j)\beta_j - \sum_{ij} \beta_i \beta_j K(x_i, x_j) \text{ , } \beta \text{ is the Eigen-vector} \qquad \textit{Equation 2.34}$$

Subject to the constraints:

$$0 < \beta_j < C \qquad \textit{Equation 2.35}$$

$$\sum_j \beta_j = 1 \ for \ j = 1, \dots, N$$ *Equation 2.36*

As previously mentioned the appropriate kernel for SVC is the Gaussian kernel with the width parameter q:

$$K(x_i, x_j) = \ e^{-q\|x_i - x_j\|^2}$$ *Equation 2.37*

We define now the distance of each image point from the center of the sphere:

$$R^2(x) = \|\Phi(x) - \alpha\|^2$$ *Equation 2.38*

Using Equation 2.28 and the definition of Mercer kernel, we have:

$$R^2(x) = K(x, x) - 2\sum_j \beta_j \, K(x_j, x) - \sum_{ij} \beta_i \beta_j \, K(x_i, x_j)$$ *Equation 2.39*

SV lies on the surface of the sphere, i.e.

$$R = \{R(x_i) | x_i \ \in \ SV\}$$ *Equation 2.40*

Vice versa, the contours that define the clusters are given by:

$$\{x | R(x) = R\}$$ *Equation 2.41*

They formed the cluster boundaries. One last step is necessary to complete the algorithm. We need to assign a cluster to each data points and label them. We observe that the path connecting two given data points belonging to different clusters must exit the hyper sphere of the feature space. Such a path must contain a point y with $R(y) > R$. This define the adjacency matrix A_{ij} between 2 points whose images $\Phi(x_i)$ and $\Phi(x_j)$ lies in or on the sphere:

$$A_{ij} = \begin{cases} 1 \ if \ for \ all \ y \ on \ the \ line \ segment \ connecting \ x_i \ and \ x_j, R(y) \leq R \\ 0 \ otherwise \end{cases}$$ *Equation 2.42*

This is better illustrated through a graphical representation:

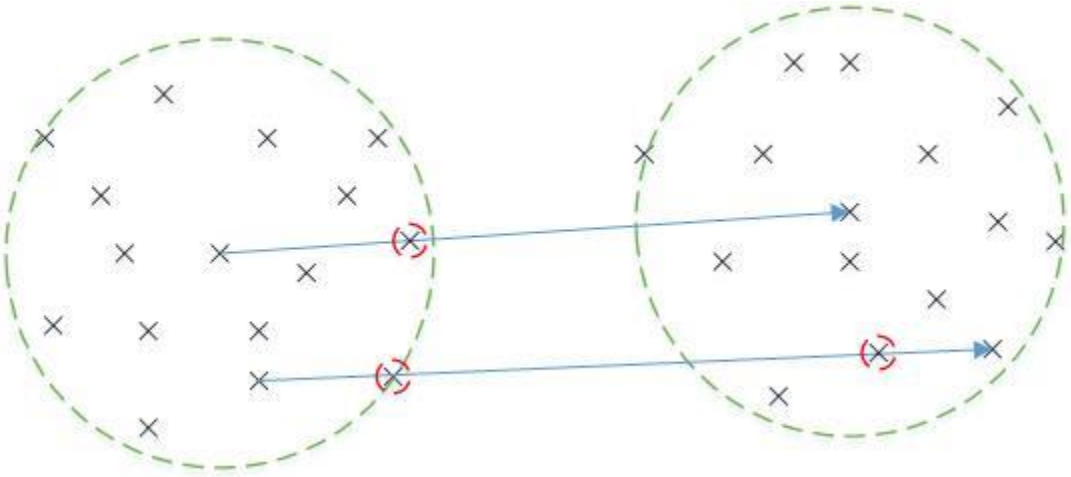


Figure 2.8: Line segment connecting data points

With this matrix, clusters are now defined as the connected components of the graph induced by A.

At this point, we make one observation here, BSVs are unclassified by this procedure. A possible solution is to assign them to the closest cluster..

The last part of this chapter will give a brief overview of two key mathematical concepts used in SVC.

2.7 Mathematical aspect

2.7.1 Quadratic Programming

Quadratic Programming (QP) [25] is a class of problems that are close to linear programming problems, with the difference that the objective function contains products of pairs of variables (called quadratic terms). As we have seen in the SVC such problems are more than abstract. Fortunately this constitutes a well-known class of problems, and the solution does exist. They belongs to a broader class of problems: Convex Programming. Although solving such problems is beyond the scope of this thesis, it is necessary to give an overview of QP here.

QP problems are usually minimization problems of the following form:

$$\begin{aligned} &minimize\ c^T x + \frac{1}{2} x^t Q x \\ &subject\ to\ Ax \geq b\ and\ x \geq 0 \end{aligned}$$

Equation 2.43

This is solved by calculating the dual problem and introducing the Lagrangian. The dual is obtained using the Karush-Kuhn-Tucker complementarity [24] conditions:

$$\begin{aligned} &maximize\ b^T y - \frac{1}{2} x^t Q x \\ &subject\ to\ A^T y + z - Qx = c\ and\ y, z \geq 0 \end{aligned}$$

Equation 2.44

2.7.2 Karush-Kuhn-Tucker complementarity conditions (KKT conditions)

Let $\lambda \in R_+^m$. The following Lagrangian function is defined as:

$$L(X,\lambda) = f(X) + \sum_{i=1}^{m} \lambda_i g_i(X) \qquad \textit{Equation 2.45}$$

where λ_i is the Lagrangian multiplier.

The KKT conditions combine two conditions: The Stationary condition and the complementary slackness condition:

To maximize L according to X we need the first order derivative to be zero:

$$\frac{\partial f}{\partial X}(X) + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i}{\partial X}(X) = 0 \; where \; \frac{\partial}{\partial X} \; is \; the \; gradient \qquad \textit{Equation 2.46}$$

The complementary slackness conditions or positivity conditions stated as:

$$\min[\lambda_i, g_i(X)] = 0, \forall i \in \{1 \dots m\} \qquad \textit{Equation 2.47}$$

Once again, solving QP is beyond the scope of this thesis. For more information we refer to [21], [24], [25], [18] and many others. The literature about this topic is very rich.

Chapter 3 - Particle Swarm Optimization

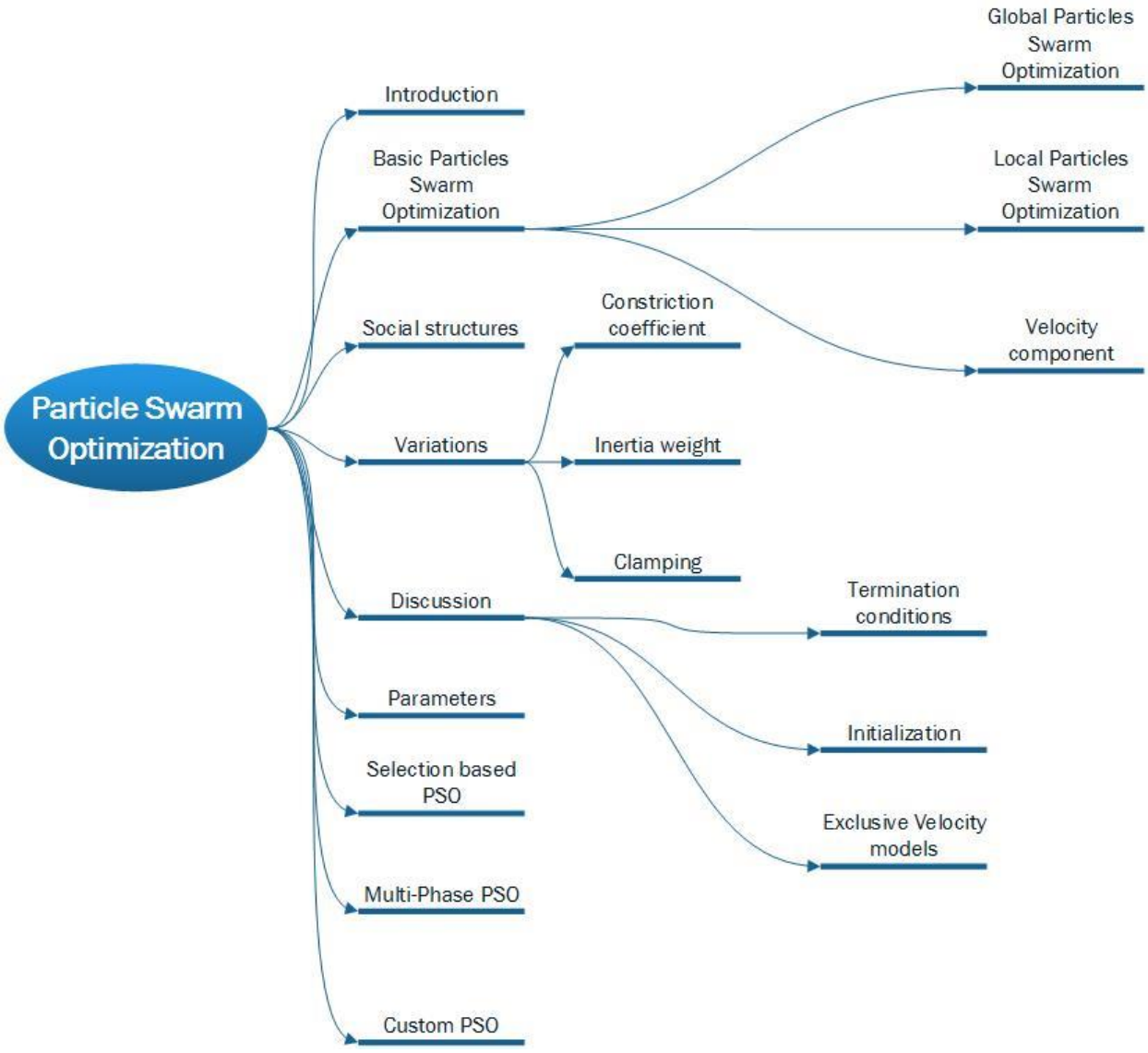


Figure 3.1: Mind map Chapter 3

3.1 Introduction

First of all, let's consider the following example. Imagine you are a bird in the middle of a flock during migration. You have to fly thousands of kilometers to reach your warm destination. You have two choices, you can either fly on your own or fly within the flock. We have all noticed the specific formation of some bird flocks and we know that it is far from being random. Flying in such formation reduces the drag. It makes it easier to fly long distances when you are in the middle of the flock. Once the birds at the edge of the flock are tired they switch with birds inside. In this way birds can migrate over very long distances. Would you prefer to fly alone or in a flock?

This simple cooperation method solves a complex problem: reducing drag in order to save energy to travel long distance. The flock can be referred to as a swarm. A swarm is composed of individuals (often mobile) that communicate directly or indirectly with each other to solve complex problems. This interaction results in distributive collective problem-solving strategies and is referred to as *Swarm Intelligence* (SI) [3]. *Computational Swarm Intelligence* (CSI) refers to the algorithmic models of such behavior [3]. The idea of creating such algorithms came from the biological study of bird flocks and ant swarms and to reproduce their behaviors on computer models. These simulations showed great ability to explore multidimensional space and quickly turned into a whole new domain of the algorithms.

We will focus on Particle Swarm Optimization [26] and some of its variations. In PSO, intelligence becomes as an emergent property from the interactions between the individuals (or particles). It means that from simple interactions one can solve complex problems. Particles follow a simple behavior: they try to emulate the success of their neighboring individuals and their own success. A PSO algorithm uses a single swarm that might be composed of many particles spread among sub-swarms. And each particle represents a potential solution to a given problem. The performance of an individual is obtained using a fitness function. This function takes the potential solution of a particle and "run" it through the current problem, and then evaluates the output.

Particles are flown into a multidimensional space, called the search space, representing all the feasible solutions of a given problem. Using the fitness function their position is updated to explore the search space and find an optimal or near optimal solution.

Let $x_i(t)$ be the position of particle i at time step t then the position at time t+1 is:

$$x_i(t+1) = x_i(t) + v_i(t+1) \qquad \textit{Equation 3.1}$$

with $v_i(t+1)$ being the velocity component of particle i at the time step t+1.

The velocity vector plays a fundamental role in the optimization process. It reflects the social information exchanged in the swarm and the own experience of a particle. The former part is referred to as the social component (SC) while the latter is referred to as the cognitive component (CC). We may say that velocity is equal to social component + cognitive component. We will present the two main PSO algorithms in the next section. They mainly differ by their social component.

3.2 Basic Particles Swarm Optimization

3.2.1 Global Particles Swarm Optimization

In global particle swarm optimization (gPSO) the neighborhood of a particle consists of the whole swarm. It means that the social component of the velocity is obtained by gathering information from all particles. We can say that the swarm is a fully connected graph of individuals. This social information is the global best position of the swarm. The SC uses the particles that achieved the best fitness in the swarm.

The velocity component for particle i at time step t+1 in gPSO is:

$$v_i(t+1) = v_i(t) + c_c r_c[y_i(t) - x_i(t)] + c_s r_s[y'(t) - x_i(t)] \quad \textit{Equation 3.2}$$

c_c is the contribution of the cognitive component and c_s the contribution of the social component. r_c and r_s are issue from a random distribution to introduce a stochastic element. $x_i(t)$ is the position of particle i at time t while $y_i(t)$ is the best personal position of particle i at time t. Finally $y_i'(t)$ is the best known position in the swarm at time t.

Let $f: R^d \to R$ be the fitness function and d the dimension of the search space. The local best position in the case of a maximization problem is obtained by:

$$y_i(t) = \begin{cases} y_i(t-1)\,, if\ f\big(y_i(t-1)\big) \leq \mathrm{f}(y_i(t)) \\ x_i(t)\,, if\ f\big(y_i(t-1)\big) > \mathrm{f}(y_i(t)) \end{cases} \quad \textit{Equation 3.3}$$

The global best position for a swarm of N particles is simply:

$$y'(t) = \max(f\big(x_1(t)\big), \dots, f\big(x_i(t)\big), \dots, f(x_N(t))) \quad \textit{Equation 3.4}$$

gPSO

Initialize a swarm of N particles
Do while until stopping condition true
    For each particles *do*
        If $f(x_i) > f(y_i)$ *do* //Set the personal best
            $y_i = x_i$
        Endif
        If $\mathrm{f}(y_i) > f(y')$ *do* //Set the global best
            $y' = y_i$
        Endif
    Endfor
    For each particles *do*
        $v_i = v_i + c_c r_c[y_i - x_i] + c_s r_s[y' - x_i]$ //Update the velocity
        $x_i = x_i + v_i$ //Update the position
    Endfor
Endwhile

Algorithm 3.1: global best PSO

The initial population of a swarm is generally a random distribution of N particles in the search space. We now present the pseudo code for gPSO:

3.2.2 Local Particle Swarm Optimization

The local particle swarm optimization (lPSO) main version of PSO differs from gPSO only in the way velocity is calculated. Instead of using the global best position to obtain the social component, this is the local best particle that is used. A neighborhood is defined for each particle. This can be done in two ways. The first method is based on the Euclidian distance, we set a distance from which a particle i belongs or not to the neighborhood of particle j. This method is computationally expensive. And spatial proximity is not required. The second approach uses particle indices.

For a swarm of size N, we define a neighborhood of size $n_s \in \{1 \dots N\}$, if $n_s = N$ we have the gPSO. The neighborhood of individual i is:

$$\mathcal{N}_i = \{y_{i-n_s}(t), y_{i-n_s+1}(t), \dots, y_i(t), \dots, y_{i+n_s-1}(t), y_{i+n_s}(t),\} \qquad \textit{Equation 3.5}$$

The local best position in the case of a maximization problem is then:

$$y_i''(t+1) = max\big(f(\mathcal{N}_i)\big) \qquad \textit{Equation 3.6}$$

The velocity is now calculated by:

$$v_i(t+1) = v_i(t) + c_c r_c[y_i(t) - x_i(t)] + c_s r_s[y_i''(t) - x_i(t)] \qquad \textit{Equation 3.7}$$

One of the advantages of using neighborhood based on particle index is that the information regarding good solutions is spread around the swarm regardless of their spatial proximity. The pseudo-code of the lPSO is given in Algorithm 3.2

IPSO

Initialize a swarm of N particles
Do while stopping condition true
    For each particles *do*
        If $f(x_i) > f(y_i)$ *do* //Set the personal best
            $y_i = x_i$
        Endif
        If $f(y_i) > f(y')$ *do* //Set the global best
            $y' = y_i$
        Endif
    Endfor
    For each particles *do*
        $v_i = v_i + c_c r_c[y_i - x_i] + c_s r_s[y_i'' - x_i]$ //Update the velocity
        $x_i = x_i + v_i$ //Update the position
    Endfor
Endwhile

Algorithm 3.2: local best PSO

3.2.3 Velocity component

The Velocity update equation is composed of three parts in every PSO variations.

- The first part consists of the momentum. This is the memory component of a particle. It represents the previous velocity of an individual. An inertia weight can be affected to this component, it balances the importance of the momentum.
- The second part is the cognitive component. This is the selfish behavior of a particle and drives a particle toward its best known personal position. A coefficient is attributed to this component to moderate its importance in the velocity update equation.
- The third and last part is the social component. As opposed to the cognitive component, it drives a particles toward the best known position of the swarm or the neighborhood of the individual. It is also affected by a coefficient.

3.3 Variations

PSO showed a great ability to solve standard optimization problems [27] and Neural Network Optimization problems [28] [29]. Their goal is to improve the convergence and the quality of the solutions found by PSO. A fundamental aspect of PSO is the trade-off of *Exploration-Exploitation* (EE trade-off) of the search space. *Exploration* is the ability of the swarm to leave its initial area and spread through the search space. *Exploitation* focusses on a promising area to refine potential solutions. There are two obvious extreme cases of the EE-tradeoff. Firstly, if a swarm favours Exploration then it might miss some good areas of the search space. Secondly, if a swarm favours Exploitation it might be trapped in local minima (or maxima depending on the problem). Within PSO the problem is addressed by the velocity update equation. In this section we present three variations of the velocity update. The velocity clamping, the inertia weight and the Constriction coefficient are discussed.

3.3.1 Clamping

The velocity update consists of three components that determine the future position of a particle. In PSO the velocity may increase quickly and the swarm might diverge completely. This means that the EE trade-off is unbalanced toward Exploration. At every update, a particle literally jumps form one position to another. To limit the growth of the velocity, the Clamping equation has been introduced. It acts as a speed limitation on the velocity updates:

$$v_i(t+1) = \begin{cases} v_i'(t+1), \text{if } v_i'(t+1) \leq V_{MAX} \\ V_{MAX}, \text{if } v_i'(t+1) > V_{MAX} \end{cases} \quad \textit{Equation 3.8}$$

With $v_i'(t+1)$ is the result using the equations Equation 3.2 and Equation 3.7. V_{MAX} is the "speed limit". This value is important because it controls the EE trade-off. Small value increases the exploitation, large value encourages exploration. The problem is to find the value for V_{MAX}, the authors of [30] used a fraction of the domain of each dimension of the search space:

$$V_{MAX,j} = \delta(x_{max,j} - x_{min,j}) \quad \textit{Equation 3.9}$$

The coefficient δ is determined using experiments.

The use of V_{MAX} restricts the step-size of each particle, but also the direction of the movement. This change of direction might encourage exploration, but might also skip the optimum. V_{MAX} does not have to be constant and can change over time. A common strategy is to start with a large value to encourage exploration and then reduce it to exploit the good region found in the first stage.

3.3.2 Inertia weight

The inertia weight is a control mechanism for the EE trade-off [31]. In the velocity update equation we have three components, and in front of the social and cognitive component there is a contribution coefficient for each. However, there is no such coefficient for the momentum. This is exactly the role of the inertia weight, it modulates the contribution of the momentum to the velocity update with w being the inertia weight:

$$v_i(t+1) = wv_i(t) + c_c r_c[y_i(t) - x_i(t)] + c_s r_s[y'(t) - x_i(t)] \quad \textit{Equation 3.10}$$

For $w \geq 1$ the swarm diverges and explores the search space at accelerating speed. Particles cannot move back to promising area. For $w < 1$ the particles decelerate until they become stationary. The optimal value for the w is problem dependent (as for the velocity clamping). However, it has been shown that w should not be under a certain limit to guarantee convergence of the swarm [32]:

$$w > \frac{1}{2}(c_1 + c_2) - 1 \quad \textit{Equation 3.11}$$

Again, the inertia weight does not have to be constant. Five different approaches have been used, random adjustment, linear decreasing, nonlinear decreasing, fuzzy adaptive inertia and increasing inertia. An issue with the inertia weight is that velocity clamping is still necessary to limit the divergence of the swarm.

3.3.3 Constriction Coefficient

The last control mechanism for the EE trade-off is the Constriction coefficient. And it is also the one we are going to use in this thesis. The idea is the same as for the inertia weight. However, the coefficient applies for the three components of the velocity update equation:

$$v_i(t+1) = \chi\left[v_i(t) + c_c r_c[y_i(t) - x_i(t)] + c_s r_s[y'(t) - x_i(t)]\right] \quad \textit{Equation 3.12}$$

With

$$\chi = \frac{2k}{\left|2 - \phi - \sqrt{\phi(\phi-4)}\right|} \quad \textit{Equation 3.13}$$

and $\phi = c_c r_c + c_s r_s$ and $\phi \geq 4$ and $k \in [0,1]$. These equations were obtained using Eigenvalue Analysis [33]. We can observe that $\chi \in [0,1]$ meaning that the velocity is reduced at each time step. There is no need for Velocity Clamping. The parameter k controls the EE trade-off, a small value implies fast convergence with local exploitation while a large value means high exploration and slow convergence of the swarm. It is interesting to start with a high value of k and progressively reduce it. This approach and the inertia weight approach are equivalent except that there is no need for velocity clamping.

3.4 Discussion

3.4.1 Exclusive velocity models

Three models have been explored by J. Kennedy [34].

- *The Cognitive-Only model*: The social component of the velocity update equation is removed. The resulting model tends to perform local search around the initial position and display poor performance in general. However, it is well suited for niching algorithms.
- *The Social-Only model*: This time the cognitive component is removed from the velocity update equation. The swarm converges faster than for the Cognitive-only model and demonstrates better result for dynamic environments [35]
- *The Selfless model*: This model is almost like the Social-Only model with one exception. In this model, the current particle is not allowed to become the best solution of its neighborhood. The performance of the selfless model is poor in dynamically changing environments, but outperforms the Social-Only model in specific cases.

3.4.2 Initialization

The first step of any PSO algorithm is to initialize the control parameters and the swarm itself. A general approach is to uniformly cover the search space, using a generator of random particle within the search space. To ensure a good coverage of the search space at initialization, Sobol and Faure sequences have been used [36] [37].The initial velocity is usually set to zero.

3.4.3 Termination conditions

Another aspect of PSO is the stopping condition. It has to satisfy two conditions. First of all, it should not stop the PSO to converge too fast, since it might find a suboptimal solution. Secondly, it should not require frequent calculation of the fitness function to minimize the number of computations needed. Five major stopping conditions have been used:

Iterations: The PSO stops when a certain number of iterations is reached. If this limit is too small, the swarm will find a suboptimal solution. If this limit is too large the PSO might make useless computations. This condition is often used with other convergence criteria as a safety stopping condition, in case the other one fails. However, during our experiment we have found this method very efficient to simply explore a search space.

Acceptable solution: The PSO stops when an "acceptable" solution has been found. A threshold is defined, and when a particle achieves a fitness within this threshold, the algorithm stops. However, this method requires a priori knowledge of the optimum, and we need to define what is acceptable. If the problem is the training of a neural network, then the optimum is usually zero because we want to minimize a measurement error. In that specific case this solution is well suited. But most problems don't require this a priori knowledge of the optimum.

No more improvements: The PSO stops when it fails to improve the fitness over a certain number of iterations. Here, if the best solution of the swarm stays stuck in a small window over a predefined number of iterations, we may consider that the PSO has converged and the solution is extracted. The main issue

with this condition is that it introduces two additional parameters to the PSO: the threshold that defines an acceptable solution, and the window of iteration.

Normalized Swarm radius close to zero: The PSO stops when the radius of the swarm becomes small. The normalized swarm radius is obtained using the following equations:

$$R_{norm} = \frac{R_{max}}{diameter(S)} \qquad \textit{Equation 3.14}$$

where,

$$R_{max} = \|x_m - y'\| \qquad \textit{Equation 3.15}$$

x_m being the particle the further away from y'. Diameter(S) is the diameter of the swarm at time step zero. This approach puts the global best at the center of the swarm. In this way, if the best position is still moving, then the radius of the swarm is not yet close enough to zero. There exists a variation of this stopping condition, this is given by the Particle Clustering Algorithm:

Particle Clustering Algorithm

Initialize Cluster $C = \{y'\}$
For about k times *do*
    Calculate the centroid of C
        $x' = \frac{\sum_{i=1, x_i \in C}^{|C|} x_i}{|C|}$
    For each particles *do*
        If $\|x_i - x'\| < \epsilon$ do
            $C \leftarrow C \cup \{x_i\}$
        Endif
    Endfor
Endfor

Algorithm 3.3 Particle Clustering Algorithm

This algorithm creates a single cluster composed of ϵ % of the Swarm. In other words, with $\epsilon = 0.9$ the PSO stops when 90% of the particles are centered at the best solution. It gives more control over the spatial disposition of the swarm to stop the search.

Small objective function slope: The PSO will stop when the slope of the objective function is close to zero. We obtain the slope at time step t by:

$$f'(t) = \frac{f\big(y'(t)\big) - f(y'(t-1))}{f(y'(t))} \qquad \textit{Equation 3.16}$$

We recognize the derived number of the objective function. If $f'(t)$ stays below a certain threshold ϵ during a set number of iterations, we consider that the swarm is not making any progress and we can stop the PSO. This solution might cause the PSO to be stuck in a local minima, i.e. if a small group of particles are stuck in a local minima, then the objective function slope might not evolve, it will cause the PSO to stop. To avoid this problem, this stopping condition can be used with the radius method.

3.5 Parameters

The parameters of PSO may vary from one version to another, but they all share a certain number of parameters. We have already discussed some of them:

The size of the swarm, a large initial population implies a large diversity and a good space coverage. However, the larger the swarm, the larger the computation per iteration is. A big swarm might also obtain an optimal solution in fewer iterations. Usually a swarm should have between 10 and 30 particles [38]. Even if good results can be obtained with smaller swarms. The optimal size will depend on the optimization problem to be solved. A smooth search space needs fewer particles than a rough one.

The number of iterations will also depend on external parameters. We have already seen the impact of a given parameter in section 3.4.3.

The neighborhood size influences the interaction between particles. A small neighborhood implies few interactions and vice versa. However, small neighborhoods tend to avoid local minima while large one exploits more. A good strategy is to increase the size of the neighborhood during the search.

We need to make a last comment on the acceleration coefficient, the social and the cognitive coefficient don't need to be constant. Smooth search spaces are well suited for a large social coefficient while rough search spaces are more suited for a large cognitive component. The adaptive acceleration coefficient has been proposed by Clerc [39]. There is a lot more to discuss about the impact of different acceleration coefficients, but in this thesis we will use only a constant one. The next parts of this chapter will present two specific types of PSO. And at the end we will introduce the algorithm that has been designed for this project.

3.6 Selection based PSO

3.6.1 Genetic Algorithm

The next PSO algorithm uses Genetic Algorithms (GA) approach [40]. This types of algorithm is inspired by the genetic evolution of chromosomes. They maintains a population of genes (equivalent of a particle in PSO). The main idea is to take this initial population, mutate it, and reproduce certain of its individuals. Mutation and reproduction are used in the biological context. We then replace a part of the initial population by the newly created genes. This section will briefly describe the major steps of any GA as:

- Parent Selection
- Crossover (or reproduction) strategies
- Mutation strategies
- Replacement strategies

3.6.1.1 Parent Selection

There are frequently used selection operators: random selection, proportional selection, tournament selection, rank-based selection, Boltzmann selection, Elitism and Hall of fame.

We will present the strategies we are going to use in the design of our custom PSO:

Random selection: each individuals has a probability of $\frac{1}{N}$ to be selected, where N is the total number of individuals. This selector doesn't use fitness information, meaning the best and the worst individuals have the same probability to be used as parents.

Proportional selection: The selection is biased toward the fittest individuals. The higher the fitness of an individual, the higher its probability to be selected is.

Tournament selection: a random group of the population is selected, the best individuals of this group is then returned. If two parents are needed, this strategy is applied twice.

Many variations exits, and there are no rules to decide which one to use. For more information we refer to A. Engelbrecht [3].

3.6.1.2 Crossover strategies

Crossover operators are divided in three categories:

- Asexual: only one parents is used for the reproduction.
- Sexual: two parents are needed for the reproduction. A noticeable point is that the same parent can be selected twice depending of the selection operator used.
- Multi-recombination: more than two parents can be used for the reproduction.

Each of these categories can be further divided, Figure 3.2 illustrates this:

- *One-point crossover*: each parents is split in two, the offspring is the combination of a part of each parent. The crossover point is randomly selected.
- *Two-point crossover*: each parents is split in three, each part is then swapped to create a new offspring. Crossover points are randomly selected.
- *Uniform crossover*: Same principle but with more than two crossover points, each part of a parent if swapped.

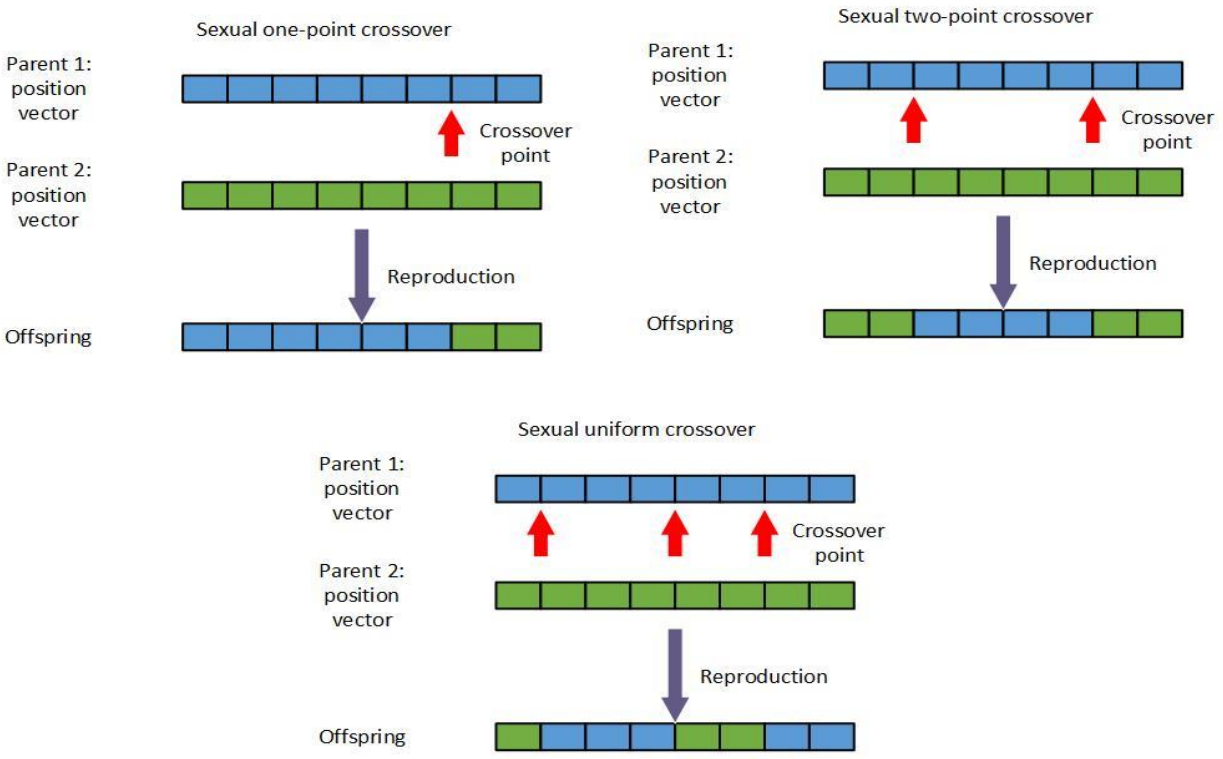


Figure 3.2: Points crossover illustration

3.6.1.3 Mutation strategies

Mutation operators are used to introduce new genetic material in the population. In other word, it introduces diversity. Along with the reproduction operators, it ensures a strong diversity. Mutation is applied at a certain probability P_m to each offspring. Mutation points are selected on the offspring. The mutation operator we are going to use is defined as:

A random value between 0 and 1 is added to the mutation points. For instance, m_i is a mutation point in an individual, then the mutated value is defined as:

$$m_i = m_i + U(0,1) \qquad \textit{Equation 3.17}$$

$U(0,1)$ being the Gaussian distribution between 0 and 1.

The next figure illustrates the mutation process.

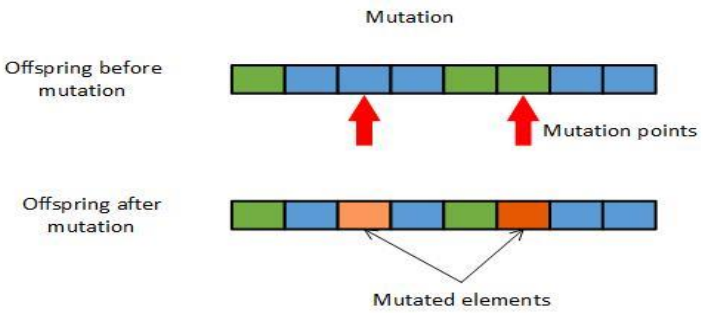


Figure 3.3: mutation operator

3.6.1.4 Replacement strategies

Once the offspring is created and mutated if necessary. It needs to be integrated in the current population. Here is a list of the mains replacement strategies:

- Replace worst, where the worst individuals are replaced by the new offspring.
- Random replacement, where individuals are randomly selected to be replaced by the offspring.
- Parent-offspring, where offspring replace their parents.
- Elitist strategy for the above strategies exists, the best individuals are protected from the replacement.

3.6.1.5 Summary

To briefly summarize, a generic genetic algorithm repeats the following steps until certain termination conditions are satisfied:

1. Select parents in the current population
2. Apply a crossover operator to generate new individuals
3. Apply a mutation operator to ensure diversity without destroying good individuals
4. Integrate the new individuals in the current population

3.6.2 Selection-based algorithm

Selection-Based PSO is one of the first version of PSO combined with Genetic Algorithm given on the next page. Algorithm 3.4 is executed before the velocity update in either local or global PSO. The memory of the low-fitness particle is not lost. Their personal best position is conserved. Even if Selection-based PSO improves the local search capabilities of PSO. The diversity is greatly reduced because half of the swarm is removed. However, this downside can be overcome by applying mutation. In other words, the worst half of the swarm is replaced by a mutated version of the top half. This maintains the diversity of the swarm. The top half generates an offspring through mutations.

The reproduction process involves the selection of the particles that will generates an offspring. Clerc [39] allowed a particle to generate a new one, kill itself or modify the inertia and acceleration coefficient. If no improvement in the neighborhood is observed, a new particle is generated in the neighborhood. With this approach, a large swarm reduces the probability to generate a particle. And a small swarm increases this probability. Another approach [41] proposed to spawn a particle in the neighborhood of the global best particle to reduce the complexity of the spawning process. The global best particle will generate an offspring if it gets stuck in a local minima. This can be detected using the stopping condition previously discussed. We will now present the Gaussian mutator introduced by Higashi and Iba [42].

Let $x_i'(t+1)$ be the position of a particle after the velocity update, and P the probability to mutate. Then for each component $j \in \{1 \dots d\}$ if $U(0,1) < P$ then the component $x_{ij}'(t+1)$ is mutated using:

$$x_{ij}(t+1) = x_{ij}'(t+1) + N(0,\sigma)x_i'(t+1) \qquad \textit{Equation 3.18}$$

where $N(0,\sigma)$ is the Gaussian distribution σ is given by:

$$\sigma = \propto \left(x_{max,j} - x_{min,j}\right), \propto \in \{0,1\} \qquad \textit{Equation 3.19}$$

Different Gaussian and Cauchy mutators have been proposed in [43], [44], [45] and [46]. Cauchy mutators are mutators based on the Cauchy distribution instead of the Gaussian.

Algorithm 3.4 presents the pseudo code for the Selection-Based PSO.

Selection-Based PSO

Initialize Cluster $C = \{y'\}$
For each particle *do*
    Select n_s particles
    Score the performance of the current particle against the n_s selected particles
Endfor
Sort the particles according to their score
Replace the worst half of the swarm by the best half without changing their personal best positions

Algorithm 3.4 Selection-Based PSO

The next section introduces the last variation of PSO.

3.7 Multi-Phase PSO

The sub-swarm has been developed by Løvberg *et al*. [47] and Al-Kazemi and Mohan [48]. Multi-phase PSO split the swarm in subgroups where each subswarm exhibit different behaviors and perform different tasks. Individuals can be allowed to migrate between groups. A common approach is to split the initial swarm in two groups and randomly assigned individuals to the groups. Then, the research strategy alternate between two phases. Firsts of all a group is in *Attraction phase,* means that particles within the group will move toward the global best position. The second phase is the *Repulsion phase* where individuals move away from the global best position.

The velocity update equation is then defined as follows:

$$v_i(t+1) = wv_i(t) + c_c x_i(t) + c_s y'(t) \qquad \textit{Equation 3.20}$$

In order to change between phases, the triplet representing the inertia weight, the cognitive and the social coefficient (w, c_c, c_s) is changed. The attraction phase pushes the particle toward the global best particle, i.e. they tend to ignore their cognitive component and follow the social one. The triplet is then: (1,-1, 1). For the repulsion phase, individuals do the opposite, they ignore the social component and move following their personal best position. The triplet is then: (1, 1, -1).

A group can change the phase after a user-specified number of iterations or when the there is no improvement of the fitness value. In order to detect an improvement or that a stagnation occurs in the fitness of a group, the termination conditions are presented in section 3.4.3. Furthermore, the velocity vector of every particles is periodically randomly reset. This procedure must be used with caution. Since it can drive a close particle away from a potential solution. To avoid this pitfall, a reset probability may be introduce. Starting with a high probability and decreasing over time, this ensures a large diversity at the beginning and then favours exploitation in later steps.

Multi-Phase PSO can exhibit more than two phases. The Life-cycle PSO [49] uses three phases. A regular PSO behavior updating the velocity using either the lPSO or the gPSO. The second phase consists of Genetic Algorithm individuals where particles reproduces with each other, mutate and finally the best offsprings are selected (Survival of the fittest). Finally, a Hill Climber phase, where a particle's position is updated only if the new position is better. They become solitary stochastic Hill-climber. In the Life-Cycle PSO the three types of individuals can coexist in the same swarm. The conditions for a change of phase are the same for the Sub-Swarm PSO. Algorithm 3.5 presents the pseudo code of the Live-Cycle algorithm:

Life-Cycle PSO

```
repeat
    For each particle do
        Evaluate Fitness
        If no improvement
            Change of the phase
        Endif
    Endfor
    For each PSO particle do
        Update Velocity Vector
        Update position
    Endfor
    For each GA individual do
        Reproduce
        Mutate
        Select the new population
    Endfor
    For each Hill Climber do
        Find the new neighboring positions
        Evaluate their fitness
        Move toward the best one with a specified probability
    Endfor
Until stopping condition is true
```

Algorithm 3.5 Life-Cycle PSO

We detailed different variations of PSO in the last section to use them as a basis for the next section. We will now explain a Custom PSO designed for this thesis that makes use of several characteristics of the previous variations.

3.8 Custom PSO

The custom PSO tries to combine different advantages of several PSO. The first step is to split the swarm in two sub swarms each in a different phase: Attraction and repulsion as introduced in 3.7 Multi-phase PSO. Each group is following the global PSO approach. From the beginning this allows a great diversity in the swarm.

At every iteration, each group updates its velocity and position. The algorithm then evaluates the probability of the worst subswarm to mutate. If the random number generated is greater than the predefined mutation probability the whole subswarm is mutated, and the phase is changed. This is the approach used in the Selection-based PSO. However, Selection-based PSO tends to reduce diversity, to avoid that we introduce the Gaussian mutation and we change the phase of the group. Finally, at later stages of the search process, the algorithm switches the behavior of the best swarm to a hill climbing behavior (defined earlier). The process stops once a predefined number of iterations has been reached. The pseudo-code of this algorithm is presented in Algorithm 3.6 Custom PSO. We choose the simple iteration condition to stop the search because the goal is to explore the search space. However, the implementation will allow us to implement different stopping conditions.

Custom PSO

```
Initialize the Swarm using uniform distribution
Split the swarm in 2 equal groups, one in attraction phase, and one in repulsion phase.
Do
    For each group do
        Update the fitness of the group //also Set local and personal best.
        For each particles do
            Update velocity using Equation 3.20
            Update position
        Endfor
        If random >probability to mutate AND current group is the worst do
            Sort the group based on fitness
            Replace the worst half by the top half (applying
            Gaussian mutation and keeping the personal best.)
            Switch of phase (Attraction/Repulsion)
        Endif
    Endfor
    If reached 80% of max number of iterations do
        Switch the best swarm to hill climbing behavior
    Endif
While max number of iteration reached
```

Algorithm 3.6 Custom PSO

We will discussed the advantages and disadvantages of this algorithm after the study cases.

Chapter 4 - Design of the system

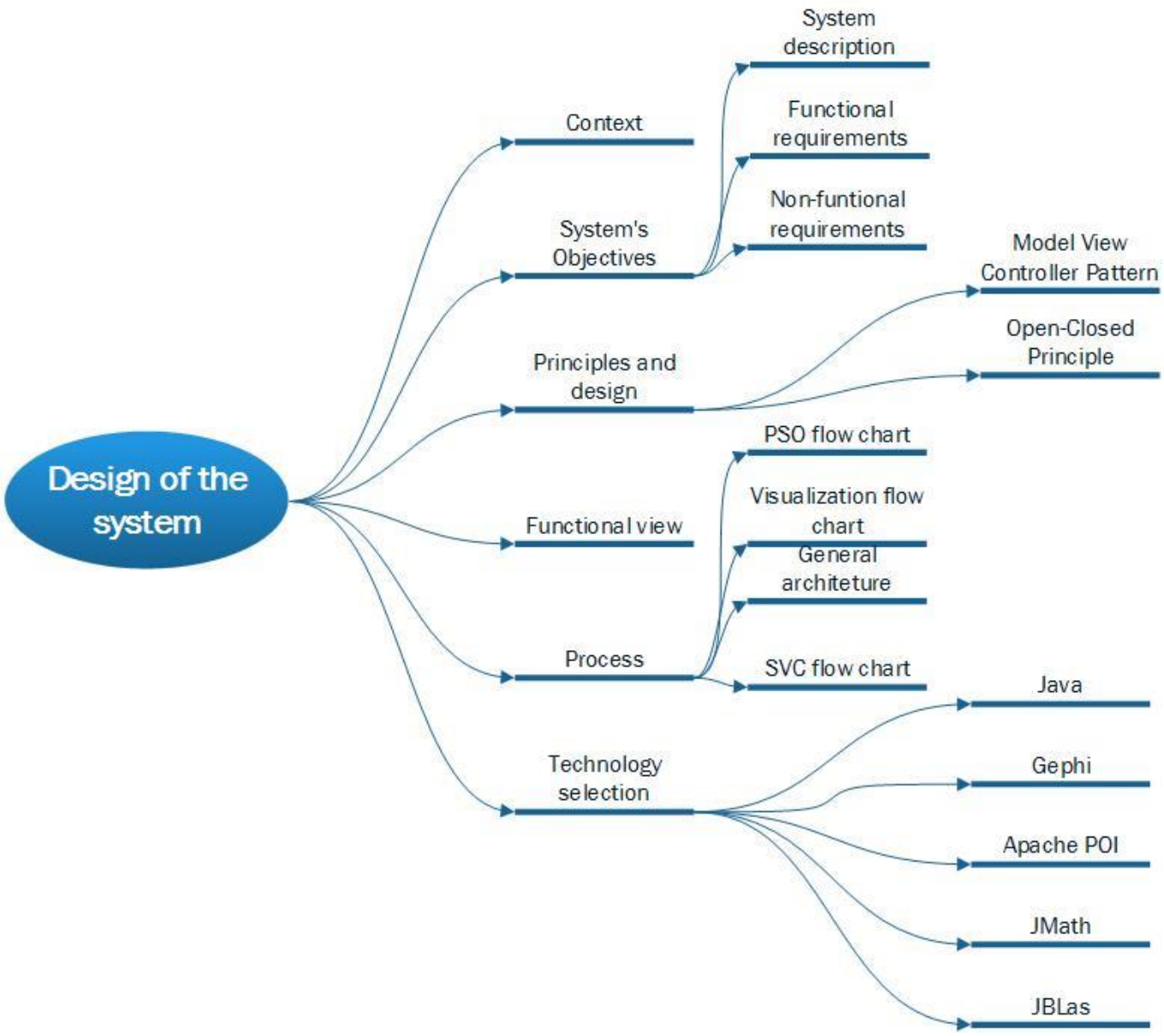


Figure 4.1: Mind map Chapter 4

4.1 Context

The software we are developing is designed to give the user the ability to run a PSO algorithm for different study cases and the SVC on different data set. Furthermore, the user needs to understand what the algorithms are doing. This means he needs to see what happened during the optimization process. It may also be necessary for him to save interesting results too. From this requirements we extracted several key features for this software:

- The architecture must be modular to easily change the data set of the SVC or the study case of the PSO.
- The user must be able to specify different parameters for each algorithms.
- The user must be able to "see" by a 2D and 3D chart the clustering or the PSO graph.
- The user must be able to export the output of the PSO or of the SVC.
- The software must be portable (can run on any computer).

We tried to follow the Scrum methodology [50] to organize the development. It helped us to select objectives for each day and week and move forward in the programming.

4.2 System Objective

The system aims to build a Particle Swarm Optimization Algorithm, a Support Vector Clustering algorithm and different study cases. It is also design to visualize the output of these algorithms. The user can save the results in different formats, and have deep views inside of the algorithm (particularly for the PSO). This means to see what the PSO is exactly doing. Furthermore, the system lets the user export the clustering results in an advanced tool or graph visualization to perform manual analysis of the results.

4.2.1 System Description

The system has to be able to perform three tasks:

1. Run PSO, SVC on a dataset
2. Visualize PSO or SVC
3. Export the data

The first task involves the following steps:

- Import a dataset into a data structure
- Setup the parameter of the desired algorithm
- Run the algorithm

The second task is composed of:

- Extract the results from the algorithm
- Translate these results to the appropriate data structure for visualization
- Display these results

The third task involves:

- Extract the results from the algorithm
- Insert these data in the appropriate file writer

- Export the data in the chosen file type.

4.2.2 Functional Requirements

4.2.2.1 Input

The data for SVC are loaded from a simple text file. Each row corresponds to a data points. The values must be convertible in a double type, only numbers are accepted. Table 4.1 illustrates the structure of a text file. As we can see, the data can be of any kind as long as they can be represented as numerical values.

Point1:Dimension1;dimension2;dimension3…;dimension N Point2:Dimension1;dimension2;dimension3…;dimension N … PointM:Dimension1;dimension2;dimension3…;dimension N

Table 4.1: Clustering Input

For the PSO, it is different, the user, in the actual version, can run only the predefined optimization problem. This limitation is due to the fact that PSO requires a fitness function that is highly problem dependent, and in the form of a java class (implementing a "Fitness" interface). The constraints linked to the fitness are in the form of a text file, where each rows represents one constraints, the total number of rows is the dimensionality of the problem (one constraint per dimension). However, we are able to implement multidimensional constraints. These constraints take the form of a java class too, and are difficult to express in a text file and extract them into the system. However, we will presents the different extensions of this software in the last chapter. For now, uni-dimensional constraints are illustrated in Table 4.5:

Line 1:[lower_bond1:upper_bond1] Line 2:[lower_bond2:upper_bond2] Line 3:[lower_bond3:upper_bond3] … Line N:[lower_bondN:upper_bondN]

Table 4.2: Uni-dimensional constraints file

4.2.2.2 Output

There are two types of output of the system:

Output of PSO:

The data that is being exported in this case are particle's position and fitness values. We save only the best known position of a particle and its associated fitness value. The final output file is an Excel file having the following structure:

Particle 1	x_{11}	…	x_{1N}	Fitness value
Particle 2	x_{21}	…	x_{2N}	Fitness value
…				
Particle d	x_{d1}	…	x_{dN}	Fitness value

Table 4.3: PSO Excel structure

Output of SVC is given by:

Regarding the SVC output, all the data are exported. This means we save the position of every data points and its corresponding cluster label. In addition the adjacency matrix is saved. The Excel file has the following structure:

Data point 1	ClusterLabel	x_{11}	…	x_{1N}
Data point 2	ClusterLabel	x_{21}	…	x_{2N}
…				
Data point d	ClusterLabel	x_{d1}	…	x_{dN}

Table 4.4: SVC Excel structure

4.2.2.3 Functionality

Functionalities for task 1: The algorithms to:

- Import the data for the chosen algorithm into a data structure
- Setup the parameters of the algorithm
- Run PSO or SVC

Functionalities for task 2: Visualization to:

- Make these results usable for the visualization process:
    - A particle will be represented as a 3D point. X coordinate is the index of the particle, Y coordinate is the iteration (time step) and Z coordinate is the fitness of particle X at the iteration Y.
    - A data point from the clustering algorithm is represented as a node in a graph, the graph is build using the adjacency matrix. The graph is then visualized using Gephi [50].(Section 4.6.4)
- Inject the results into the appropriate visualization tool
- Programmatically extract the results of each algorithm

Functionalities for task 3: Exporting to:

- Extract the results of the algorithm
- Use a file writer to export the result in an Excel file.

4.2.3 Nonfunctional requirements

- Flexibility: The data used for the clustering algorithm comes from any type of database, the only limitation is that they have to be expressed in terms of multi-dimensional data points. On the other hand, PSO is a lot more restricted. The uni-dimensional constraints are easily put into a text file. However, it was difficult to explicitly represent the fitness function and multi-dimensional constraint in a text file. We choose to restrict the PSO to the predefined use cases and integrate a more advanced solution at a later stage in the project.
- Usability: The software needs to have a graphical user interface to maximize its readability. The interface has to be both functional and intuitive to allow anyone to run the algorithm. It must be user friendly.

- Portability: The software needs to be platform independent. In this way it can be run on Windows, Mac OSX, Linux etc…without recompiling the code. The Java programming language was therefore a perfect choice.
- Adaptability: Many versions of PSO exists, many visualization techniques too. The design of the software is adaptable and will let any developer modify the source code easily and implement different versions. The detailed architecture of the software is explained in the next section of this chapter.

4.3 Principles and design

4.3.1 Model View Controller Pattern

The model view controller (MVC) pattern [52] is like any other design pattern, it is designed to enable all interactive applications to clearly separate between the different components of the architecture. The necessary functions of an application are regrouped under three categories:

- The model (data model)
- The view (UI)
- The controller (handles event, and synchronization)

The model contains the logic of the application, in our case, the implementation of the PSO, the SVC, and the visualization. It does not communicate with the view.

The view is what the user will see and use, it handles the user events such as hover[4], mouse click, text box etc…). It send the user's request to the controller.

The controller receives requests from the View and sends them to the model which executes them. It can also inform the view of any changes (unexpected events. Figure 4.2 illustrates this:

[4] A hover is the action of dragging the cursor over an element of the interface.

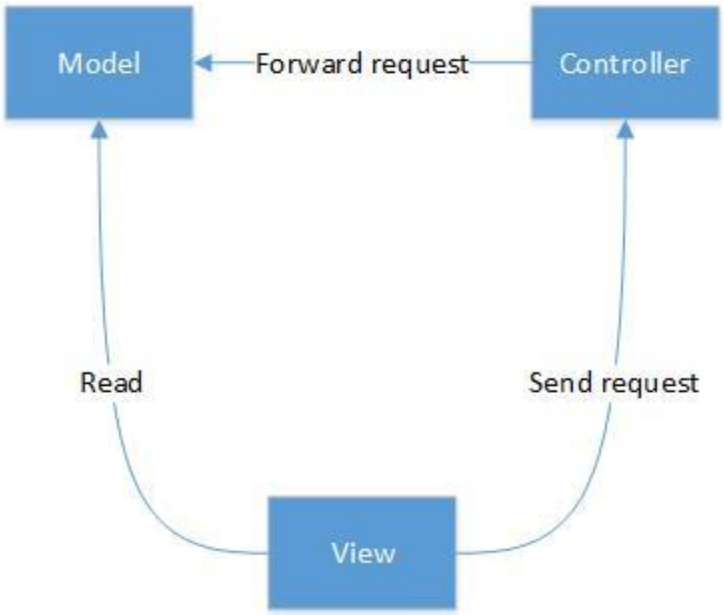


Figure 4.2: MVC pattern

In our software, every algorithm has a class that monitors every other class and interfaces needed to run the algorithm and also sends information to the view. It is at the same time a model and a controller. However, there are several algorithms, so instead of having one main controller, and a model for each algorithm the controller is split between the different algorithms. The view consists of a GUI application. Depending on the requirement, it could be a text based interface. However, in our case it was important to have a visual feedback, therefore the GUI is a good solution.

4.3.2 Open-Closed Principle

The Open-closed principle has been largely applied in this project too, it stands for: “Open for extension, close for modification” [53].

This principle states that an entity should allow the modification of its behavior without changing its source code. In other word a service should rely on abstraction and not on implementation details. In Figure 4.3.A we can see that the “PSO” class depends on the implementation of the “AttractionRepusionVelocity” class. This would work if we were sure that no other implementation of the velocity equation would be used. However, in Swarm Optimization there are many different strategies to compute the velocity equation and this design made their implementation complicated. By applying the Open-Closed principle we break the dependency across the implementation and rely on the interface which does not contain any code, but just signature of a “Velocity” type class.

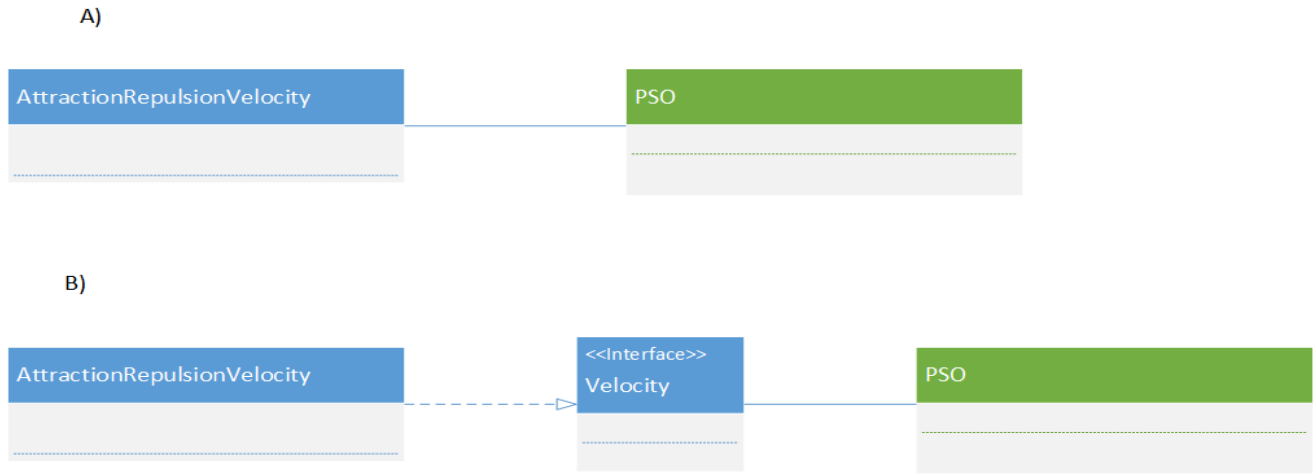


Figure 4.3: Open-Closed Principle

In this program, every study case is implemented by a "Fitness" interface allowing to just plug the proper fitness function and constraints into the PSO and run it. To make this even easier, we instantiate the class "PSOExperiment" which is composed of a fitness function and a set of constraints. The PSO then just needs this class to be able to execute an optimization problem. Figure 4.4 shows the connection between the main parts of the system. The dashed arrows represents "interface realization" relation, as we can see, the PSO does not depend on the problem it is solving, but rather on interfaces.

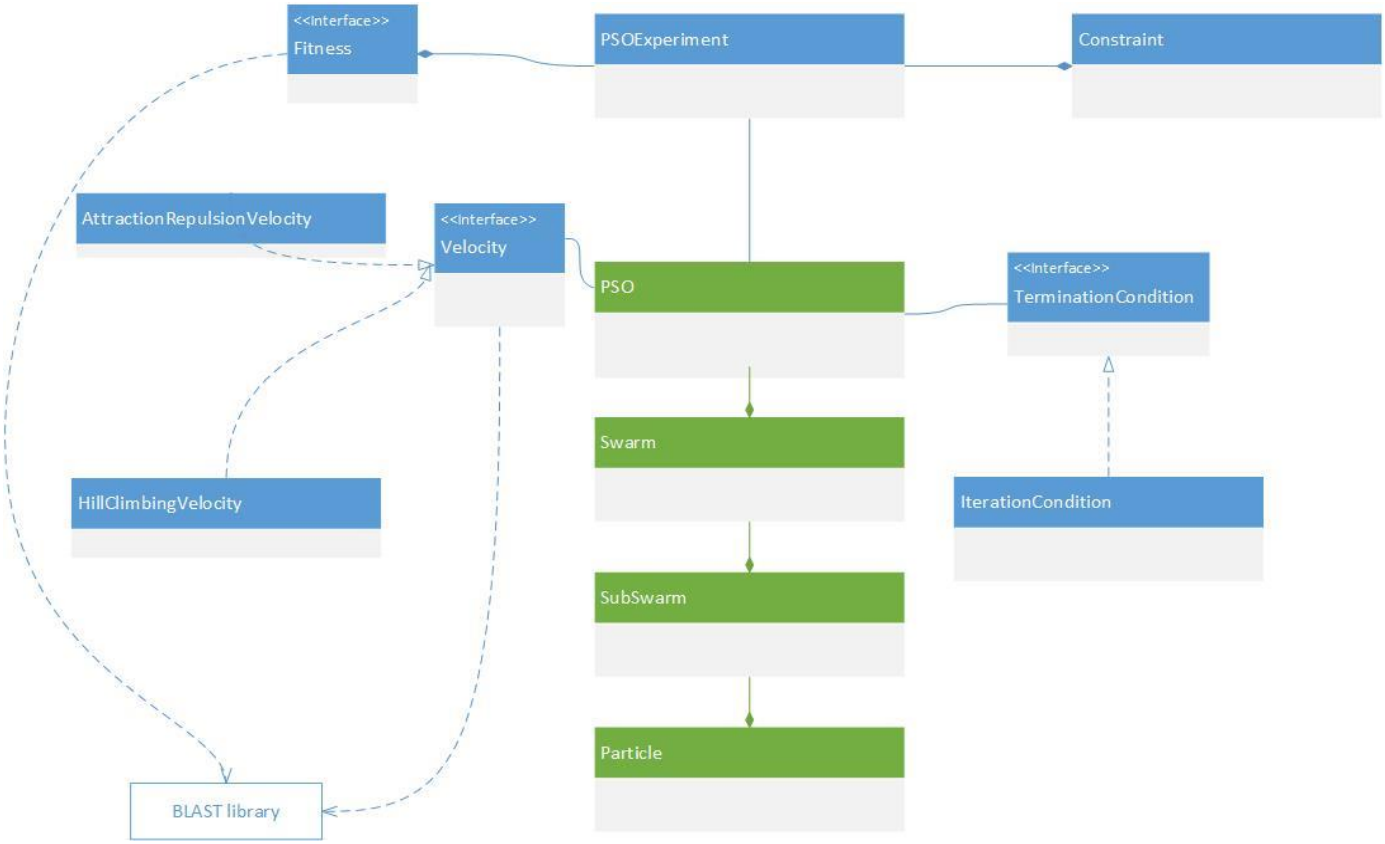


Figure 4.4: PSO class diagram

4.4 Functional view

The description of the system leads to the creation of the use cases diagram (Figure 4.5). This diagram summarizes all the interactions we want to integrate in the UI. However, it is not an exhaustive list. The software is susceptible to evolve even after this thesis. But it gives a good overview of the capabilities of the program and was an efficient tool to set objectives for the development.

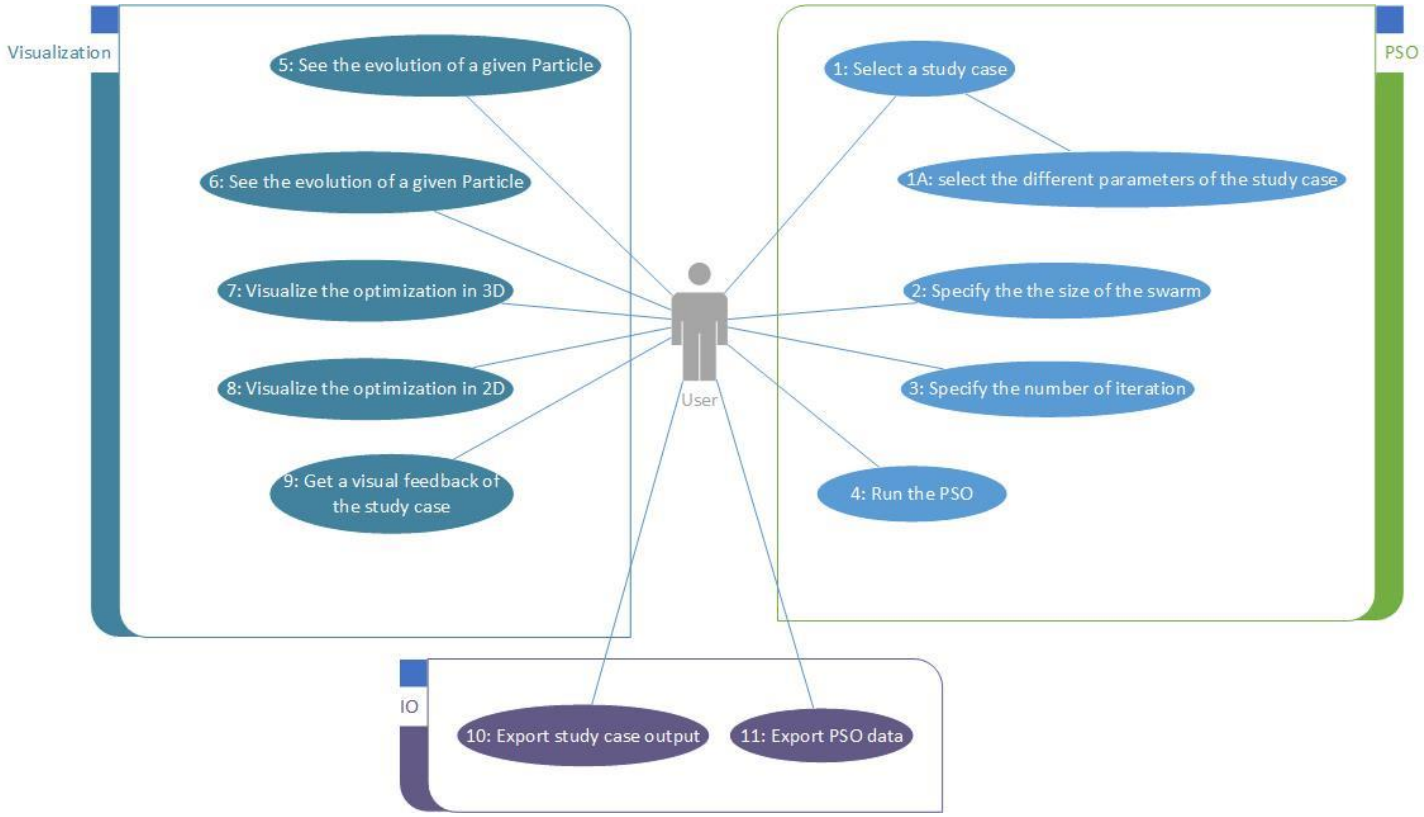


Figure 4.5: Use cases Diagram

We want the user to run the PSO through different study cases. Once the user has selected an algorithm and successfully run it, she/he can visualize what the algorithm did, and save the data. We tried to precisely define the possible user interactions and summarized them in Figure 4.5. The user needs to be able to select a study case, and specifies the corresponding parameters. She/he has to specify the size of the swarm and the number of iterations as a termination condition and finally run the PSO.

The next set of possible actions is related to visualization. It is important to let the user “see” the swarm through different scales. This means that he should be able to visualize the evolution of a single particle, a subswarm, or the whole swarm. And get also a visual feedback of a specific study case if it is relevant.

The last group of actions concerns the possibility to export the results of an experiment.

4.5 Process

In this section we will discuss the implementation of the PSO and the SVC. Instead of giving snapshots of the source code, we use flowcharts to explain the process of each algorithms.

4.5.1 PSO flow chart

The first algorithm we are going to present is the Particle Swarm Optimization. Figure 4.6 illustrates the flow chart of the algorithm.

The first step on every PSO process is to load the fitness function and the different constraints. Once this is done, the swarm is initialized. It means that the different parameters (initial size and number of iterations) are loaded and the particles are randomly generated within the search space using a Gaussian distribution. Once a particle is generated, their fitness value is evaluated for the first time, and its velocity vector is set to zero (particles are static when generated, they have no inertia yet). The next step is to iterate through all the particles of every subswarm and update the fitness, the velocity and then the position in this order. Then the probability to mutate a particle is evaluated and if it exceeds a predefined threshold it will mutate using a Gaussian mutator. However, the mutation keeps the best known position of the particle. In this way there is no loss of information. The evaluation of the hill climbing follows. We decided to activate the hill climbing behavior once we reached 80% of the maximum number of iterations. At this point in the process, one of the subswarm started to converge. The exploration is not necessary anymore, and we can try to move faster toward the best possible solution. Finally, once the maximum number of iterations is reached all the particles stop to move and the data are saved for further process. At this point the user can visualize the results and save them.

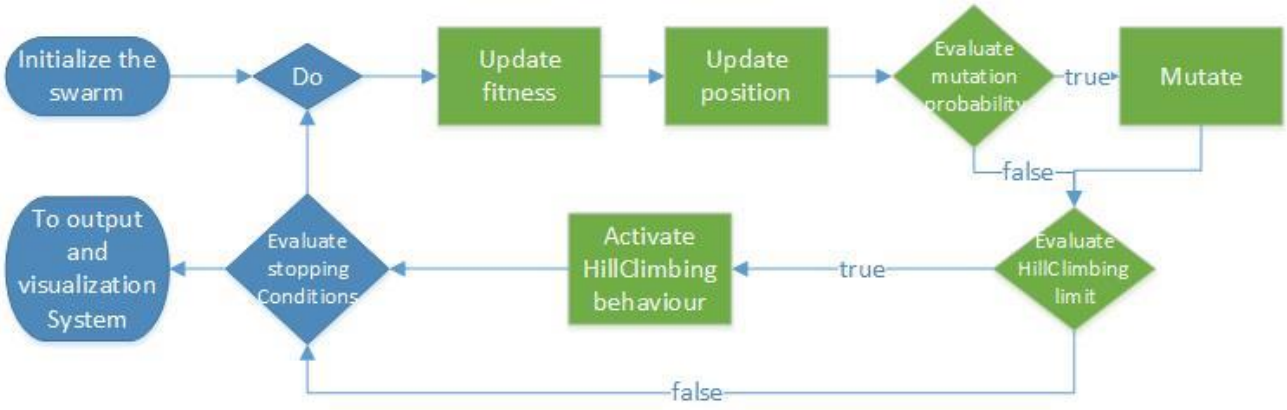


Figure 4.6: PSO flow chart

In addition, Figure 4.4 shows the class diagram of the PSO. As we can see, the PSO relies mainly on three interfaces: Velocity, Fitness and Termination Condition. We decided to abstract the concept of termination condition because as explained in 3.4.3 there are several possibilities to stop the optimization process even though we are using only a simple limit on the iteration. Further development will require different stopping criteria.

4.5.2 SVC flow chart

The Support Vector Kernel Clustering is a more straight forward algorithm even though the concept behind is more complex to apprehend.

Here, the first step is to load the data set, the data points are loaded into a matrix of double type, one row representing a single data points. Then the Kernel function is loaded, we will use only a Gaussian Kernel because Polynomial Kernel is proved to give poor results for SVC. Once the Kernel and the data are

set, the dual problem is formulated and solved using a quadratic solver or the PSO in order to compute the Lagrangian multiplier. The next step is to identify the Support Vectors and the Bounded Support Vector. This let us compute the radius of the hyper sphere in the feature space and generates the adjacency matrix. Now the points can be assigned to a cluster by finding the strongly connected component in the graph induced by the adjacency matrix. At this stage, the clustering process is done. The adjacency matrix can be visualized using the Gephi library [50] and the results may be exported.

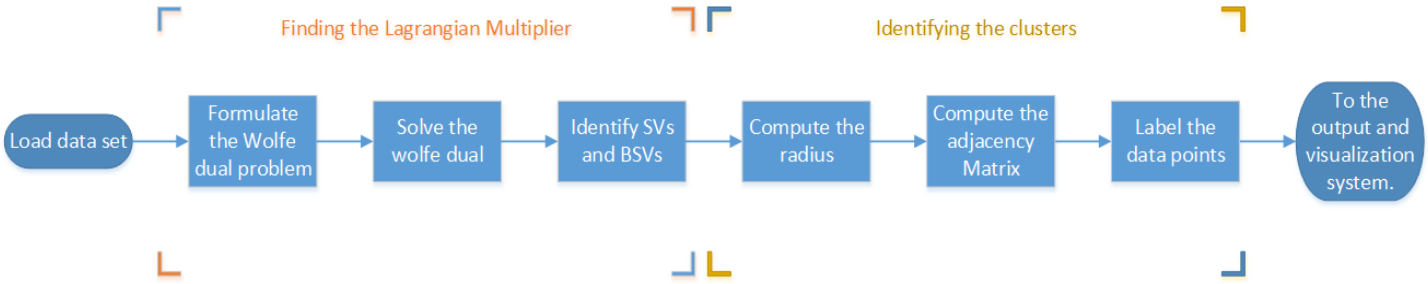


Figure 4.7: SVC flow chart

Figure 4.8 shows the class diagram of the SVC implementation. The "Open-closed" principle is not as obvious as in the PSO. Nevertheless, we can see that the algorithm relies on two major abstractions. First for the Mercer's Kernel to have the possibility to implement the others types of kernels (section 2.5) and second for the Quadratic Optimization Problem solver. In this way we have the option to use different QP solvers of the PSO and compare their results (e.g. to compute a fitness function).

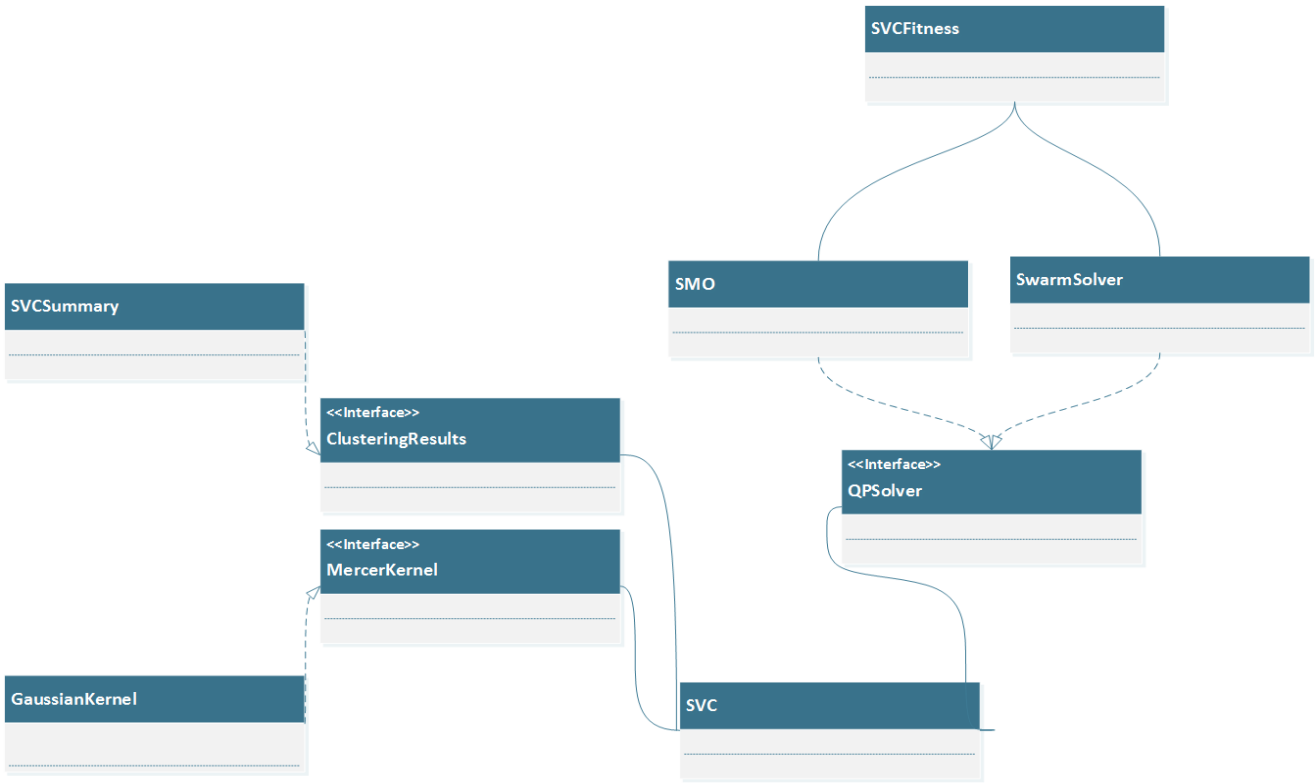


Figure 4.8: SVC class Diagram

4.5.3 Visualization flow chart

The visualization process is straight forward. The first important step is to use the appropriate visualization tool for the PSO and the SVC. In the case of PSO, the user has the choice to visualize the evolution of the swarm, a specific subswarm or even a single particle. He can do that in two-dimensions for all of the elements, or in three-dimensions for the Swarm and the sub swarms. Once the user has selected what she/he wants to visualize, the selected data are processed to fit the library for two or three dimension charts. In the case of SVC visualization, the adjacency matrix is extracted from the algorithm. The corresponding graph is built and exported as a gefx[5] file. This file can now be opened with Gephi to analyze it. We decided to use this approach to avoid the expensive and complex dimension reduction that occurs in almost every visualization technique for clustering. Both PSO and SVC results can be exported as excel files. The clustering results are stored simply as data points and their corresponding cluster labels. However, it was more difficult to know exactly what to save in the PSO. A lot of data is generated and maybe keeping only the best solution is not what the user always wants. After meeting with our collaborator Ryan Hughes at PricewaterhouseCoopers [53], it appears that to have good representation of the fitness structure, it is necessary to save the best positions of every particles in the swarm and their corresponding fitness values. This way enables the user to see if there might be more than one region of high fitness value in the search space. Figure 4.9 shows the visualization process:

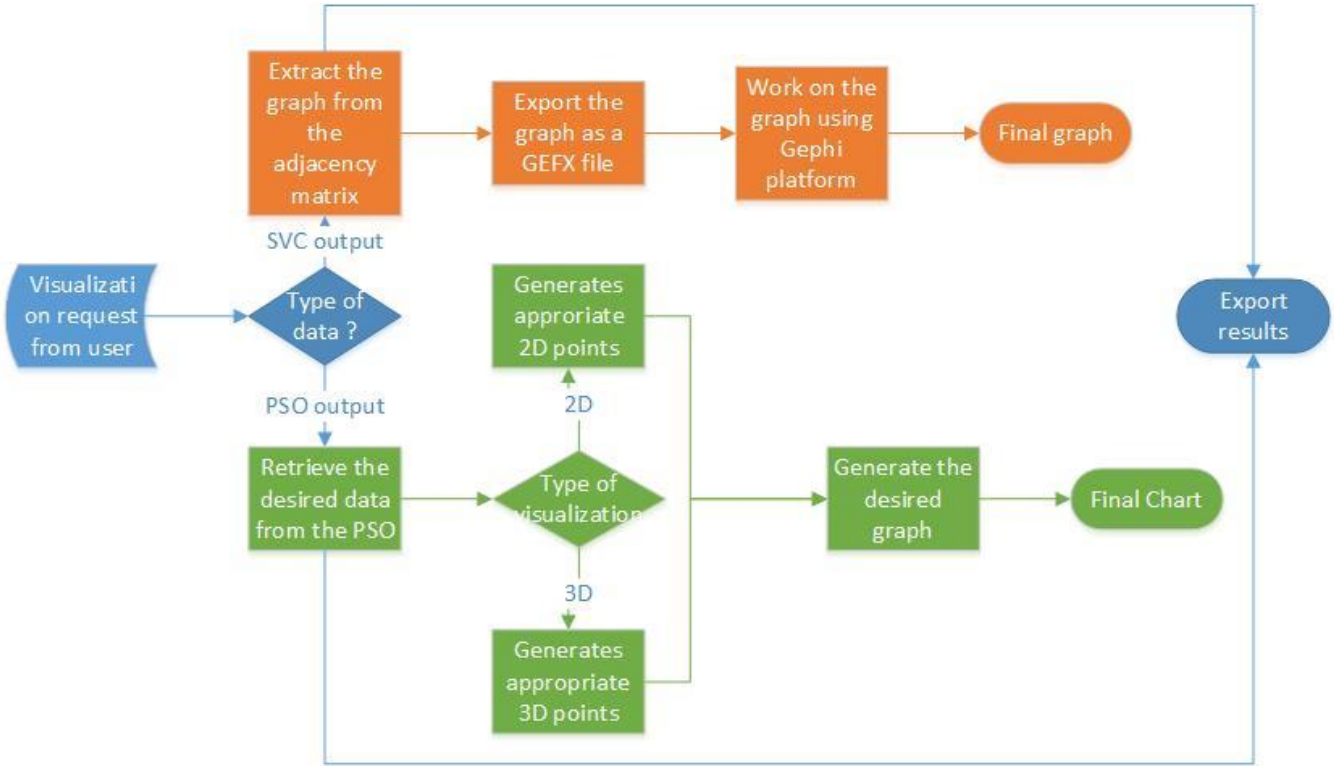


Figure 4.9: Visualization flow chart

[5] Gefx file is a specific file type for the Gephi platform to store and manipulate graph

Figure 4.10 shows the class diagram for the visualization process. The class “SIC” is the GUI, because the visualization starts from the GUI in this software. The “Open-closed” principle is not applied here but we decided to regroup all the different types of charts one class. However, each “Particle” has a method to return its data from the optimization process. Hence, the subswarm also has a method to extract these data. In this way, the process of extracting the data is straight forward and gives a lot of flexibility to display any kind of chart. The visualization class can process these data to generate the appropriate plot. It is easy to programmatically add new types of charts within the limits of the library. We choose JMathPlot mainly for its “easy-to-use” feature. The different technologies will be presented in the last section of this chapter.

The GUI also uses a TextWriter and an ExcelWriter class. These classes contain a method to export data in a text file or in an Excel file. They are used to save the data from an experiment.

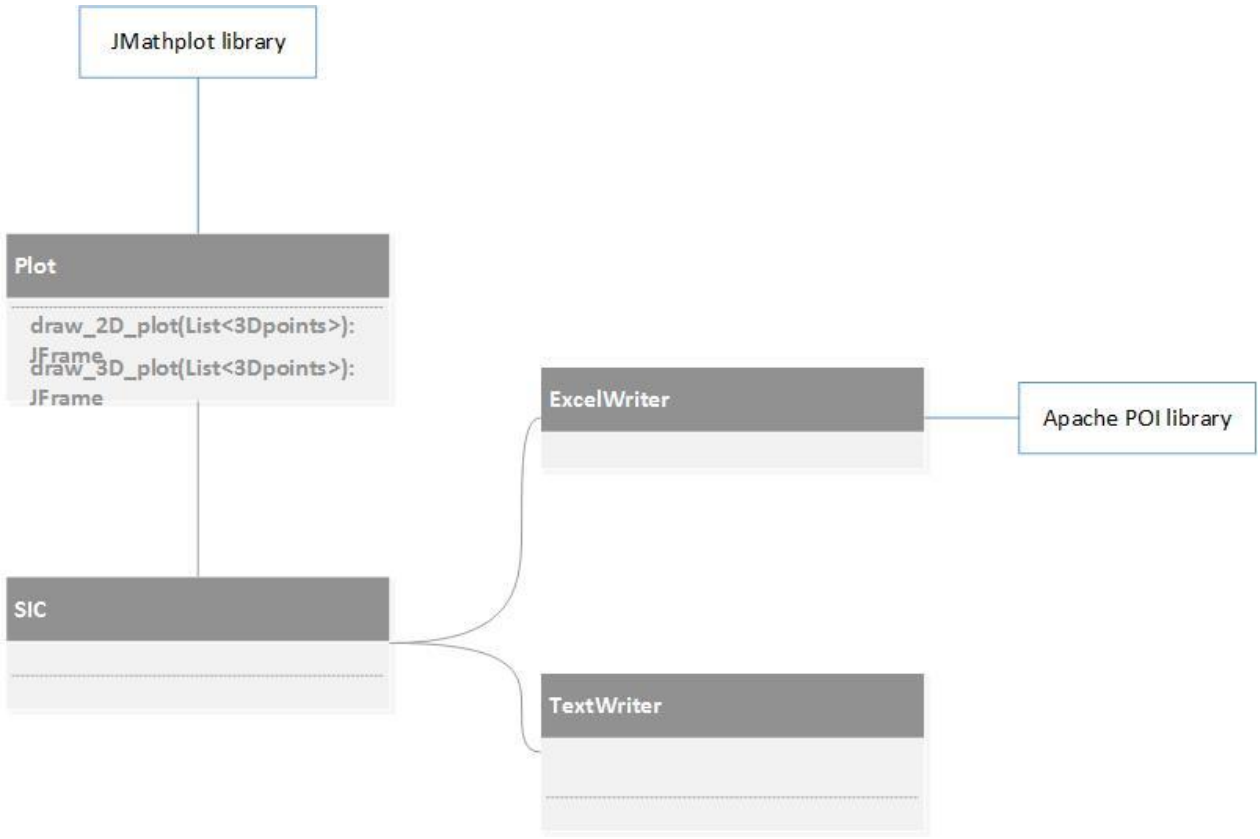


Figure 4.10: Visualization class diagram

4.5.4 General architecture

Figure 4.11 shows the general overview of the software. The GUI is related to the PSOController to define the parameter of the experiment and uses PSOExperiment to run the swarm. A PSOExperiment is simply a set of constraints and a fitness function. The constraints determines the search space for the swarm. While the fitness function evaluates the position of a particle. This fitness function defines the optimization problem. The MovieController helps to define more precisely the experiment to run. But this is specific for this study case. The PSOController can retrieve all the data from the swarm and forward

them to the GUI. The GUI decides whether to expert this data in a file or to visualize them depending of the user's requests.

On a programmatic level, the code is separate in several group of classes. Each group is coherent, this means that they fulfil a specific task such as running the optimization algorithm, dealing with the user interface, the visualization system, exporting the results and finally one for each study case. All the classes of a group is inside a package.

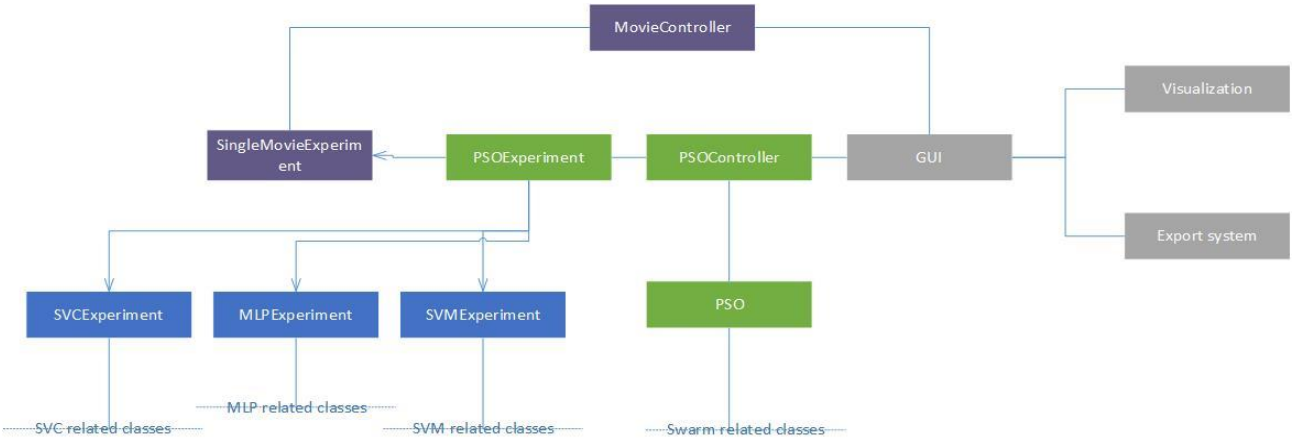


Figure 4.11: General architecture

4.6 Technology selection

This section will briefly describes the different technologies we used during the development.

4.6.1 Java

First of all, Java has been selected has the programming language because it is an Object Oriented programming langue, it is portable and also from our good experience with it, from earlier projects. Anothers attracting feature is that there are a wide variety of libraries in Java. An additional feature is that we will also be able to integrate Netbeans Rich-Client Platform for the GUI or any other platforms (Spring for instance).

4.6.2 JBLAS

JBLAS is an open source library for fast linear algebra. It is based on BLAS and LAPACK industrial standard for matrix computations [54]. JBLAS possesses a small, but active community and is easy to use.

4.6.3 JMath

JMath [55] is a set of independent packages to fit engineering and scientific computing needs. It is easy to integrate, to modify and to extend. JMath is composed of three packages. JMathArray for linear algebra for double arrays. JMathPlot is designed for plot in 2D and 3D. Finally, JMathIO is for binary and ASCII input/ouput of double arrays.

4.6.4 Gephi

Gephi [50] is a freeware for graph visualization and manipulation. As for all these libraries we choose them based on how active they are, the community is probably the most active of all of them. We choose Gephi to visualize the clustering process because it is powerful enough to let the user highlights any relevant aspect of the graph. However, Gephi was not easy to learn, but the website provides all the necessary tutorials and the source code is well documented.

4.6.5 Apache POI

The last library is Apache POI [56]. This library is designed to read and write into Microsoft Office type files from the Java programming language. As part of the Apache community, the documentation is almost exhaustive, with a lot of tutorials. Moreover, people have been very reactive to help us in understanding this large library.

Table 4.5 summarizes the different libraries we have used.

Library	**Community size**	**Community Activity**	
JBLAS	Small	Active	Linear algebra
JMath	Big	Very active	scientific plot 3d and 2d
Gephi	Very big	Very active	Graph visualization and manipulation
Apache POI	Very big	Very active	Microsoft office connector

Table 4.5: Libraries summary

Chapter 5 - User Interface

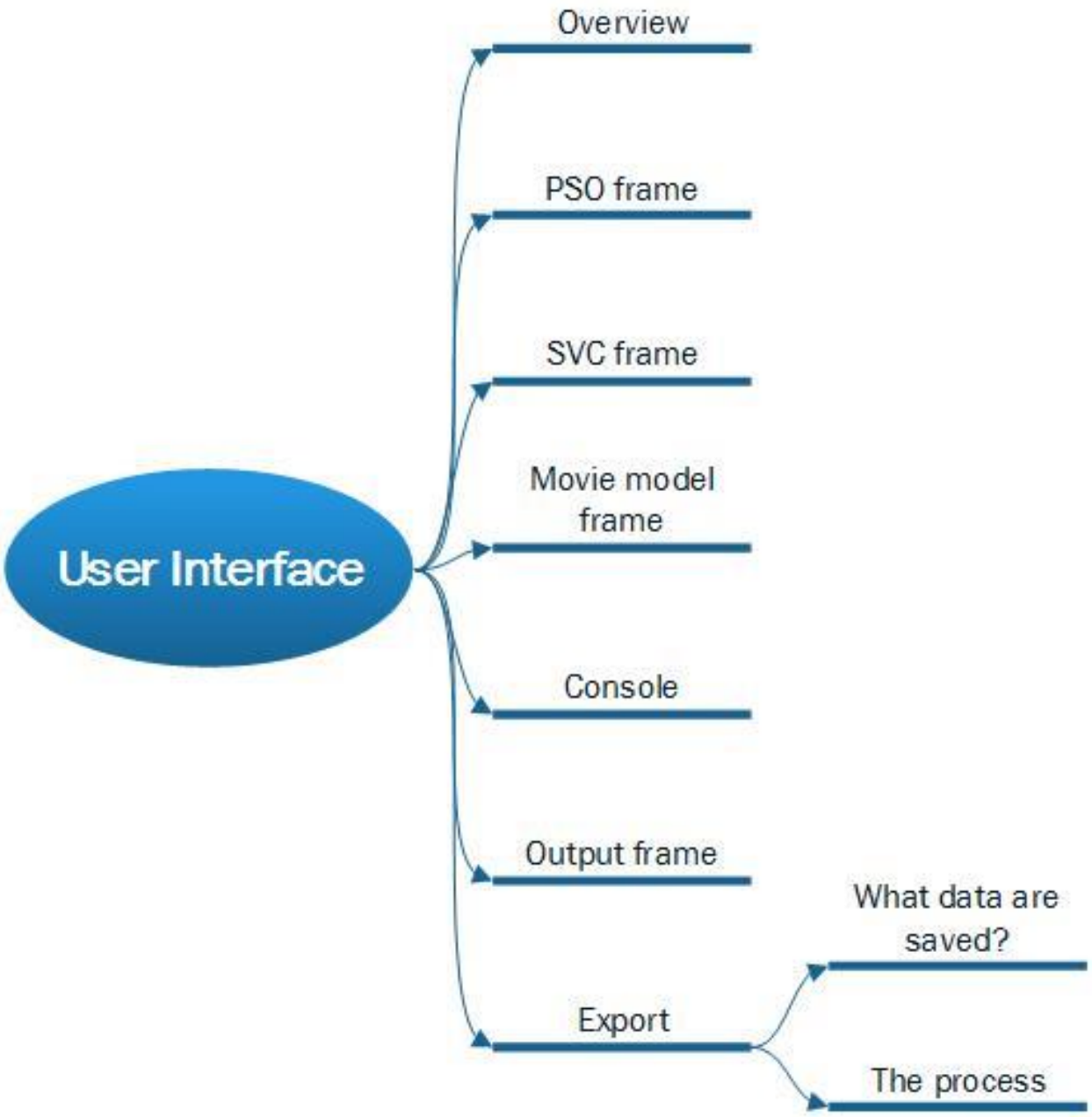


Figure 5.1: Mind map Chapter 5

5.1 Overview

5.1.1 General interface

The design of the user interface is critical, and can drastically influence the user's experience. We decided to use a Graphical User Interface, based on the framed concept of Swing. Throughout this chapter we will describe the different frames of the user interface. The GUI is split in four different frames: a console which allows to display various information regarding the current experiment and the state of the system. A visualization frame, tailored to visualize as much information as possible regarding the Swarm Optimization Process. A third frame is dedicated to the configuration of the PSO and the problem to be solved. Finally, the last frame is used to configure the desired PSO problem.

Here is a picture of the User Interface:

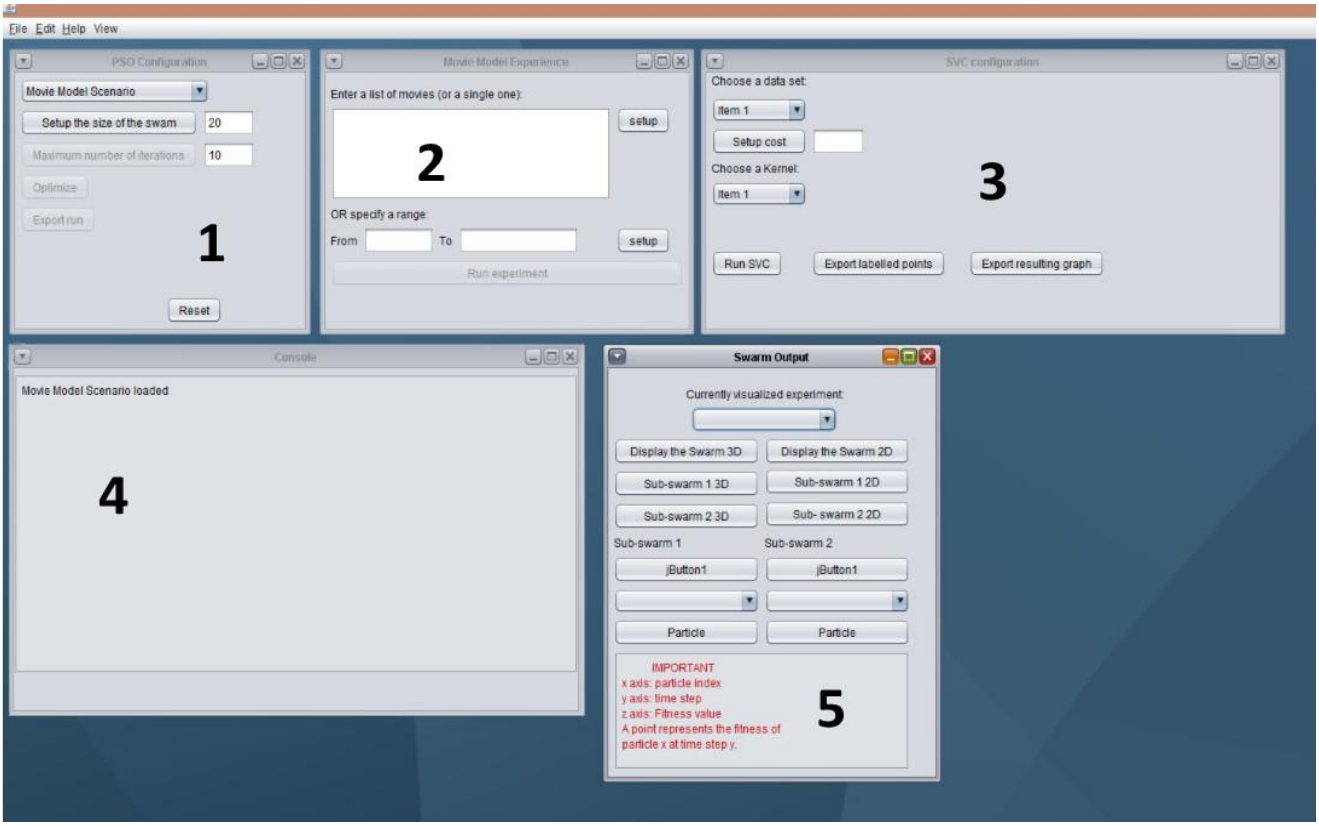


Figure 5.2: General GUI

In Figure 5.2 Label 2 and 3 are two illustrations of the configuration pane of user-selected problem. Number 2 is dedicated to the movie model case. Number 3 is for the Support Vector Clustering case.

Every frames can be closed, minimized or maximized depending of the user's needs without loss of information. I.e. if by mistakes the configuration frame is closed, it can be reopened through the "view" menu. This will restore the frame in its previous state keeping the information already selected.

5.1.2 Different frames

5.2 PSO frame

This frame is fundamental to the software as it control the Swarm. As we can see in Figure 5.3, it is composed of a Drop-down menu to select the study case to run. Once selected the corresponding configuration pane will be open for study case related parameters. This drop-down menu is followed by two text field to specify the size of the swarm and the maximum number of iterations of the algorithm. Once everything is filled in the "optimize" button "run" the algorithm, display the output frame and unlock the export button. The reset button reinitialize all the parameters to run a different experience. We tried to minimize the number of parameters the user can play with, to keep the interface as simple as possible. However, this design choice came at the expense of less control over the algorithm. Through our experimentation, it turns out to be enough parameters. On a later version, we will choose between advanced and simple configuration frame.

We will talk in more details about the export function in section 5.7.

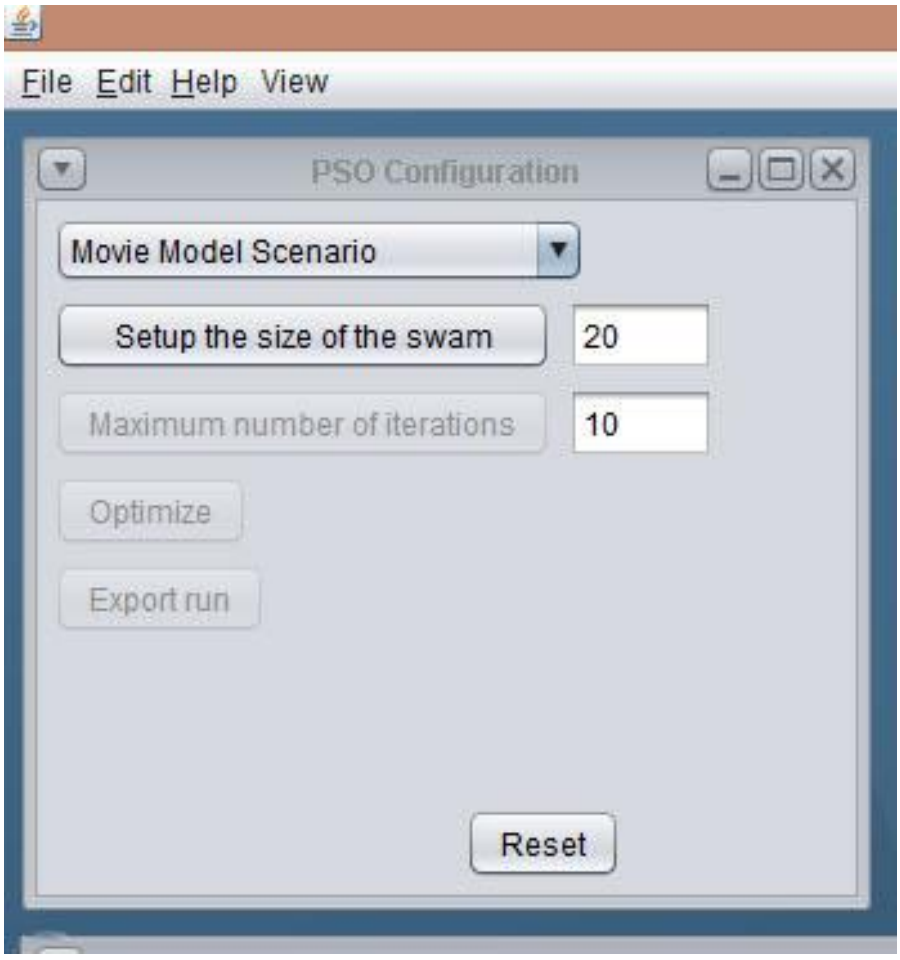


Figure 5.3: PSO frame

5.3 SVC frame

This interface let the user choose a data set to run the clustering algorithm. The text field of the cost refers to the upper bound on the Eigenvalue (see section 2.6). It is also possible to choose different Kernel functions (see section 2.5). All these parameters are sufficient to configure a Support Vector Clustering problem and to solve it. The three last button allow us to run the algorithm, and save the output in two different ways. Firstly, as spreadsheet of data points labelled by their assigned cluster. And secondly, a graph file that can be processed using the Gephi software [50] to do the visualization on the produced results.

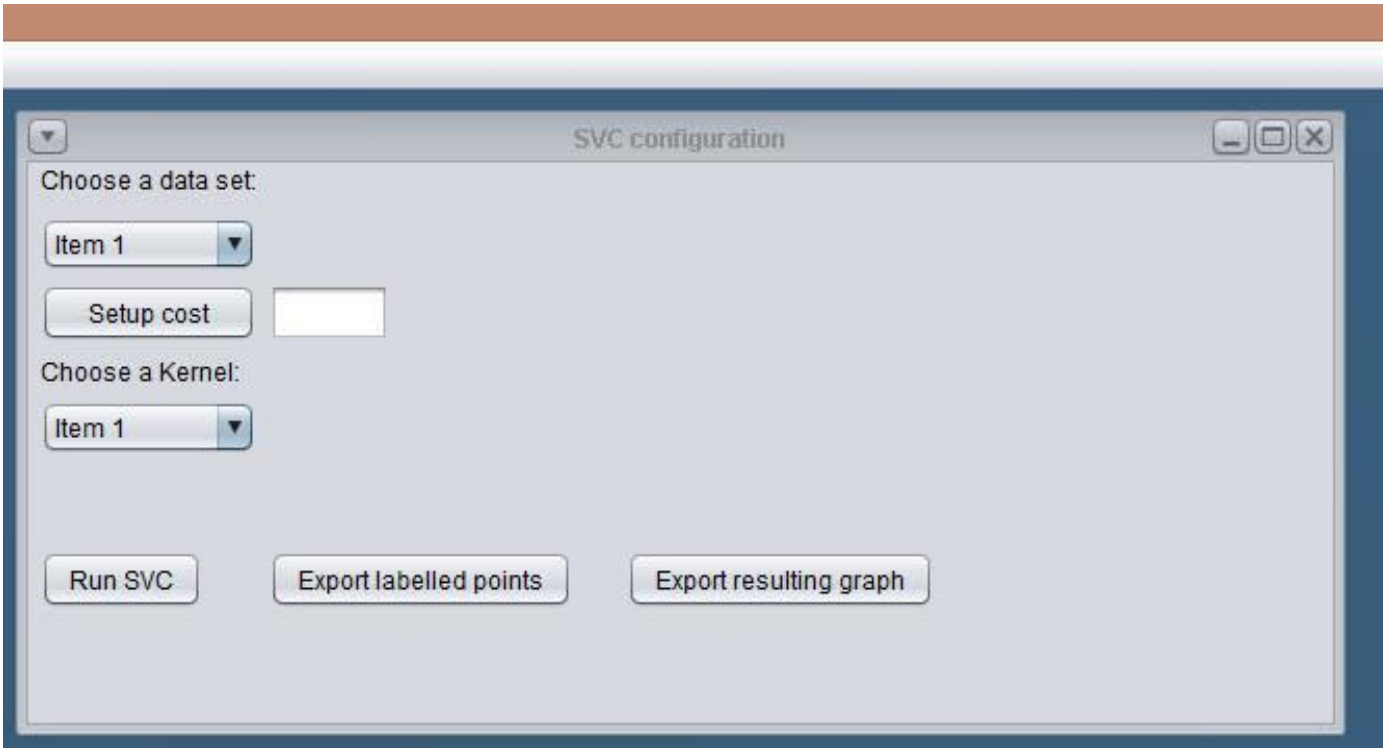


Figure 5.4: SVC frame

5.4 Movie model frame

The movie model frame, does not need a lot of option. As we will see in the next chapter this model uses a database of 154 movies, indexed by numbers ranging from 571 to 725. Here the user can target one single movie, a serie of movies (non consecutive indexed) or simply a range with the first and last index. The program will then automatically run all the movies between these two indexes. However, running these types of experiments are time consuming. This model is not optimized in its current state of development. We will explain this in more details later.

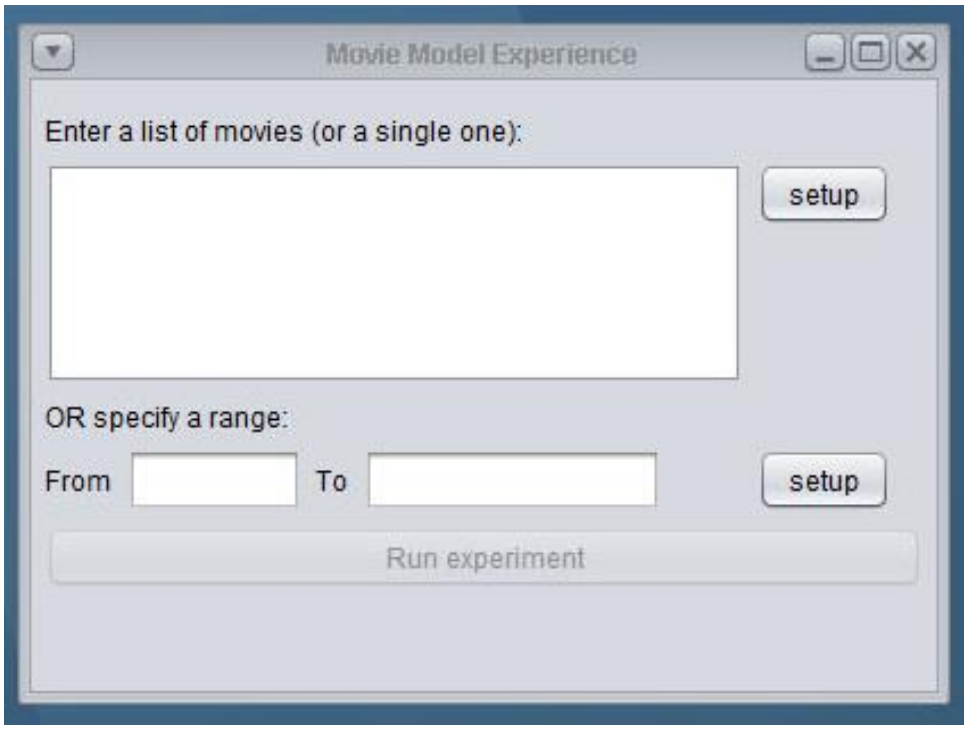


Figure 5.5: Movie model frame

5.5 Console

The console Is purely informative and gives information about the state of the system as the user configures an experiment. It has no other purpose. However, it was prooved to be very useful.

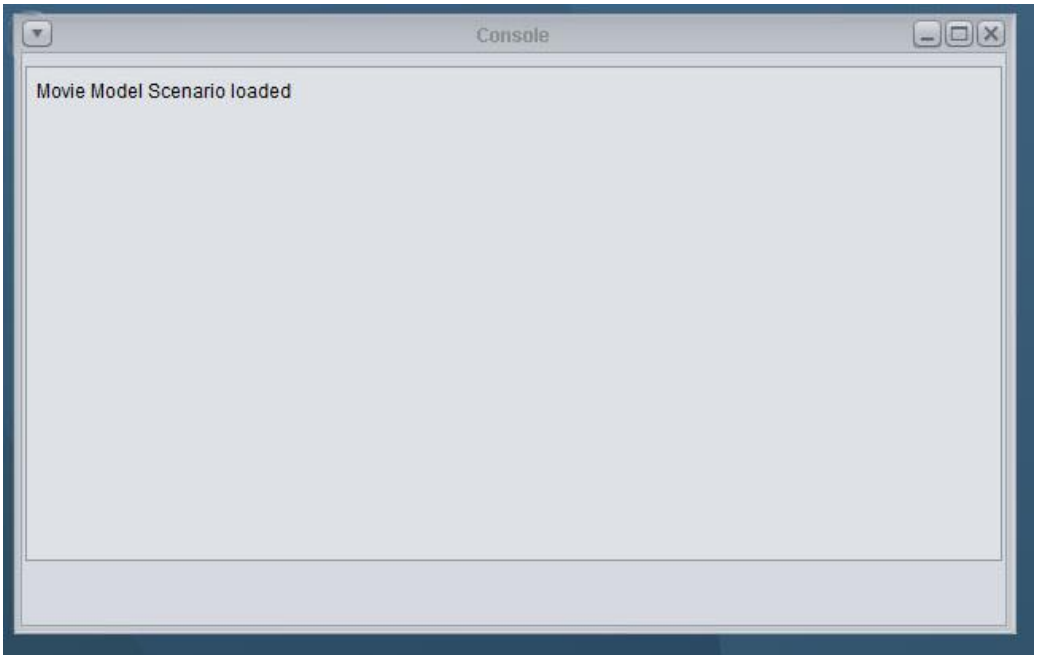


Figure 5.6: Console

5.6 Output frame

This frame has more functions, it is available only when an experiment is done. The drop down menu allows the user to select an experiment to visualize. They are stored in the order they are done. The system keeps track of everything that the swarm does during every experiment. It gives us the possibility to fully explore the swarm after the optimization process terminates. As we can see, it is possible to visualize either in three dimensions or in two dimensions the evolution of the swarm or a subswarm.

This frame gives another level of information of the swarm. Indeed, we can look at the evolution of a single particle during the process or directly select the best particle of the desired subswarm. The last part at the bottom of the frame is a simple note to explain how to read the three dimensional chart as the framework we used did not allow any legend on the axis of the chart.

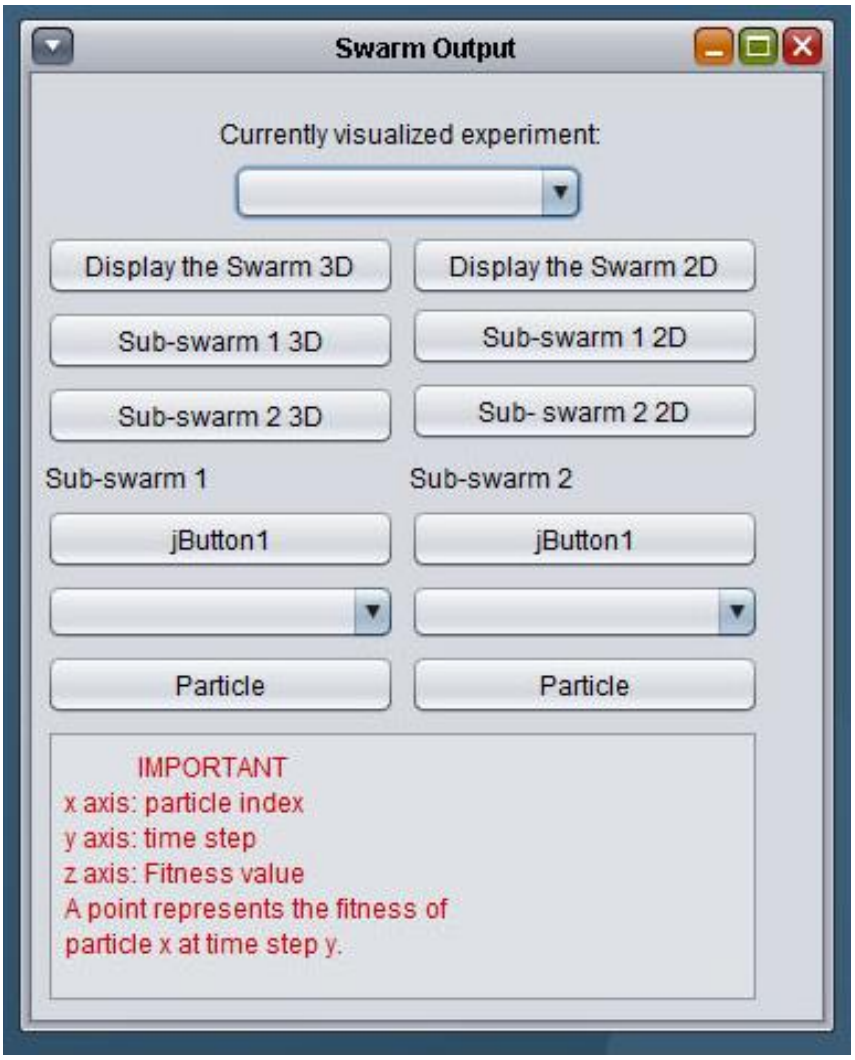


Figure 5.7: Output

If the user finds a chart particularly interesting, he can save it by simply clicking on ‘it and select the appropriate option. Even if it is a three dimensional chart from the program he can setup the view of the chart in the desired view and take a snapshot of it. Figure 5.8 is an example of a chart generated by the

algorithm. The z-axis represents the fitness, the x-axis is the time and finally the y-axis is the index of the particles.

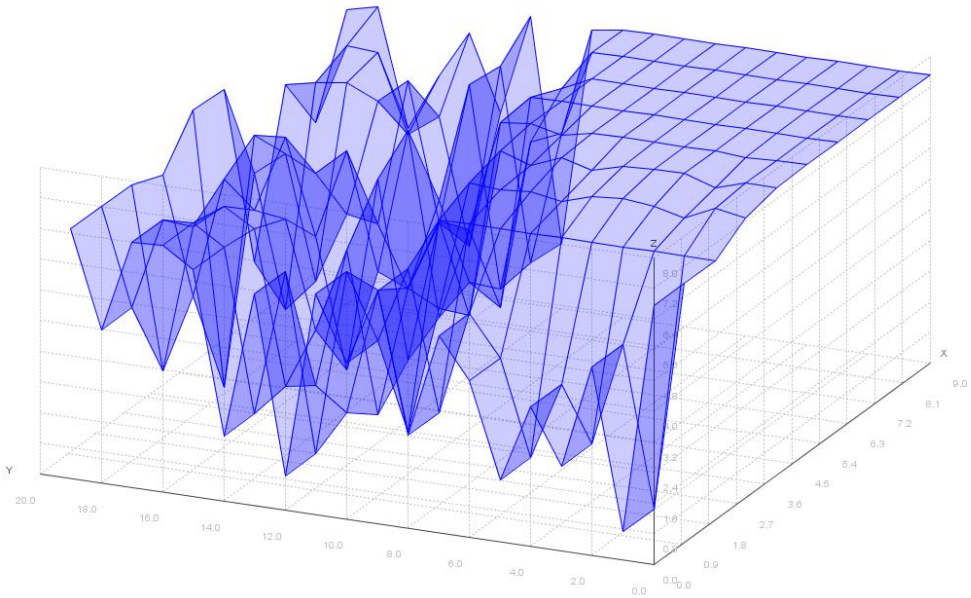


Figure 5.8: A 3D chart, description in the text above.

Figure 5.9 shows a two dimensional chart generated by the algorithm. The x- axis is the time, the y-axis represents the fitness. Each particle corresponds to one color.

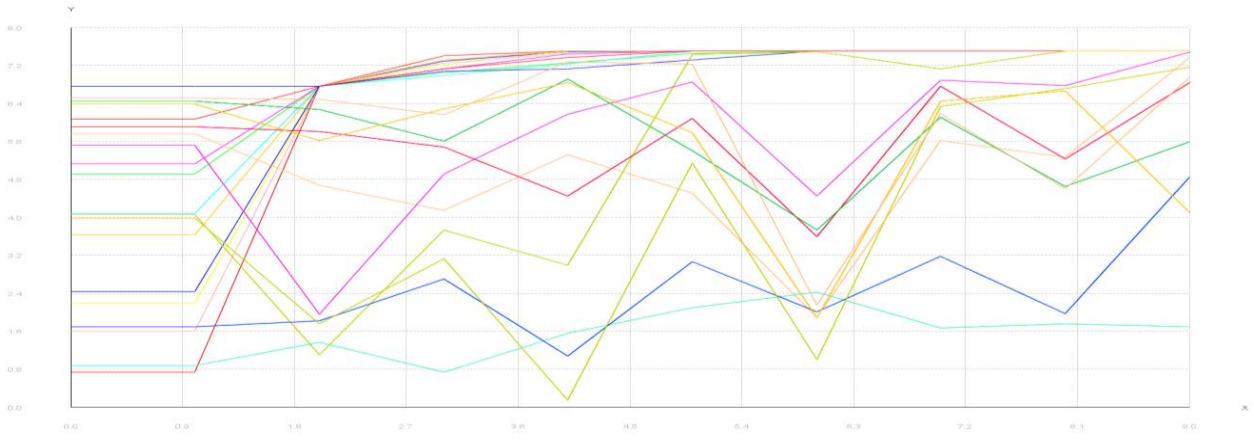
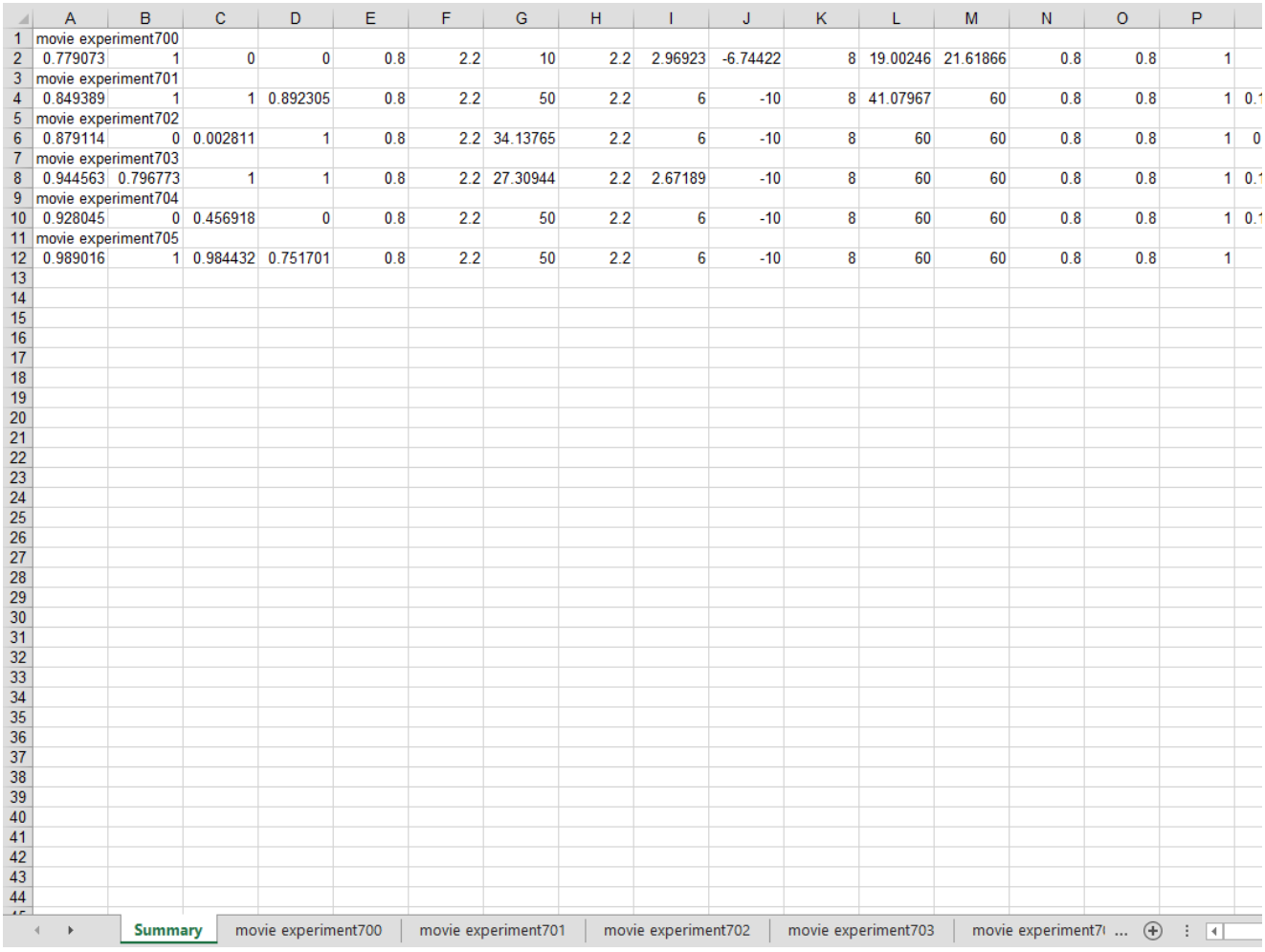


Figure 5.9: A 2D chart, description in the text above.

5.7 Export

5.7.1 Saving criterion

It did not make any sense to save all the data because most of them are useful only for the visualization part. Nevertheless, it was necessary to export more than just the final solution. For every experiment we decided to save the final position of every particle at the end of the optimization process. The reason behind this choice is simple. Some particle other than the best particle might have a high fitness and be located in a very different position than the best particle. This suboptimal particle might be worth keeping for further analysis on the user side. It might not be the case, but we decided to give the user this possibility (e.g. it could highlight unexpected acceptable solution). In addition, a summary table is created keeping track of the final solution of every experiments. Figure 5.10 shows a final excel file. We can see at the bottom the summary table and the spreadsheet created for every experiment.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	movie experiment700															
2	0.779073	1	0	0	0.8	2.2	10	2.2	2.96923	-6.74422	8	19.00246	21.61866	0.8	0.8	1
3	movie experiment701															
4	0.849389	1	1	0.892305	0.8	2.2	50	2.2	6	-10	8	41.07967	60	0.8	0.8	1
5	movie experiment702															
6	0.879114	0	0.002811	1	0.8	2.2	34.13765	2.2	6	-10	8	60	60	0.8	0.8	1
7	movie experiment703															
8	0.944563	0.796773	1	1	0.8	2.2	27.30944	2.2	2.67189	-10	8	60	60	0.8	0.8	1
9	movie experiment704															
10	0.928045	0	0.456918	0	0.8	2.2	50	2.2	6	-10	8	60	60	0.8	0.8	1
11	movie experiment705															
12	0.989016	1	0.984432	0.751701	0.8	2.2	50	2.2	6	-10	8	60	60	0.8	0.8	1

Figure 5.10: The Excel structure of the output file

5.7.2 The process

The export button needs a few explanations on its action. When the user did several experiments with the swarm he can export the results to an Excel workbook. Once clicked, the export button opens a simple file chooser to select the location where the user wants to save his data and with the desired name for the file. After this, another algorithm starts to save the important data. To be precise the system is saving every experiments in a list. This list is sent to the output system. Algorithm 5.1 explains the process to save the results of the experiments.

Output algorithm

Create a new spreadsheet ***summary***
For *each experiment* ***exp*** **do**
    Create a new spreadsheet ***e***
    Export ***exp*** in ***e***
    Export the final solution of ***exp*** in ***summary***
Endfor
Add all the spreadsheet to the workbook
Write the workbook to the desired location.

Algorithm 5.1: Output algorithm

Chapter 6 - Study case: The movie model

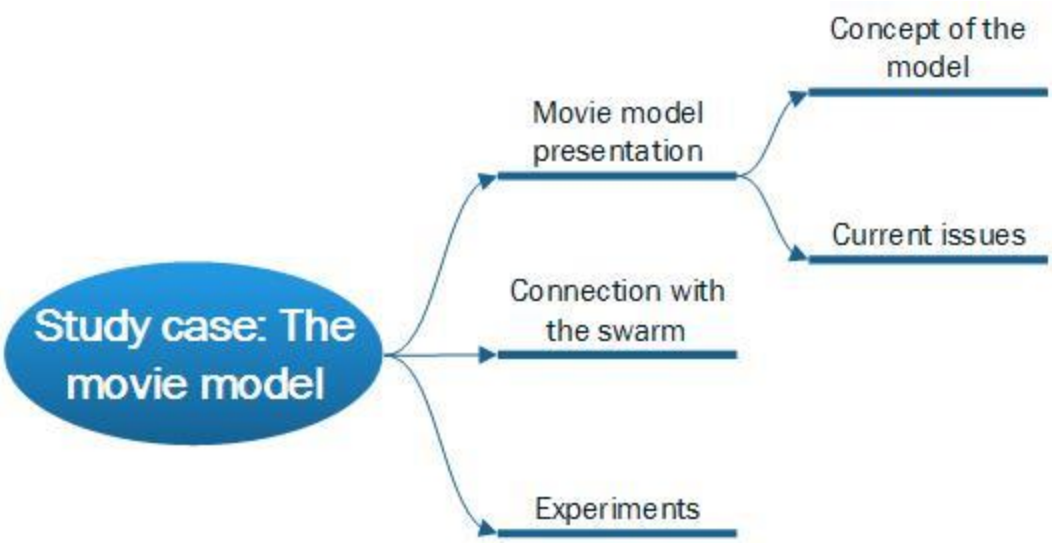


Figure 6.1: Mind map Chapter 6

Now that the system is fully described, we can present the first study case. This chapter is about the movie model. Firstly we need to introduce the movie model and explain how it works.

6.1 Movie model presentation

6.1.1 Concept of the model

This model was designed by Christopher Ryan Hughes in his master thesis from 2012 [57]. It is designed to study the impact of release strategies on the diffusion of motion-picture movies at the US domestic box-office. It captures consumer choice as a behavioral process accounting for the movie's intrinsic attributes and various other parameters (word-of-mouth, network effect, etc…). The model estimates weekly box-office receipts for a database of 154 movies. It tries to answer two main questions:

- *What is the optimal release date for a movie given marketplace competition?*
- *What distribution strategy will maximize a movie's box-office revenue given its intrinsic attributes and the behavioral characteristics of consumers?*

The model is based on the usage of different states of consumers. It means that a consumer can be in six mutually exclusive behavioral states:

1. Undecided: The consumer doesn't know the movie is playing or is undecided whether he is going to see it or not.
2. Intender: The consumer has decided that he is going to see the movie.
3. Rejecter: The consumer has decided that he is not going to see the movie.
4. Positive Spreader: The consumer is actively spreading positive word-of-mouth.
5. Negative Spreader: The consumer is actively spreading negative word-of-mouth.
6. Inactive: The consumer has seen the movie, but is no longer spreading word-of-mouth.

The state transition is described in the following state transition chart:

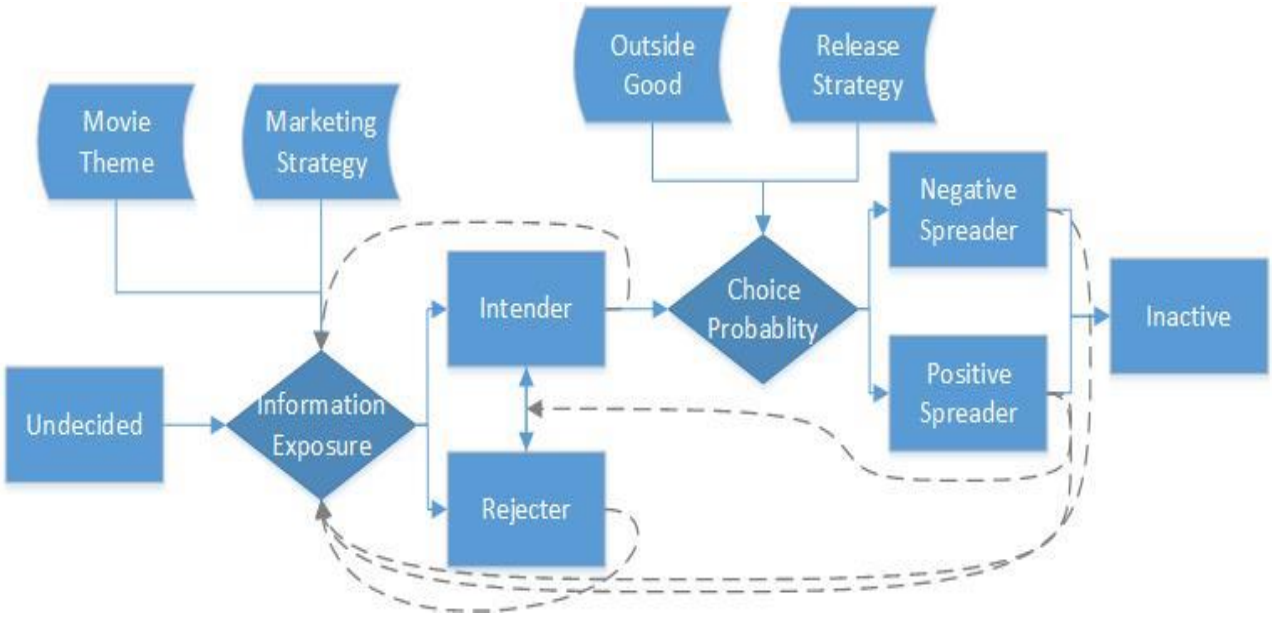


Figure 6.2: State transition diagram

This transition is ruled by equations and influenced by a set of parameters. The state transition equations are outside the scope of this thesis. The parameters that influence these states are relevant to us because it is this one that PSO is going to optimize. They are separated into four different categories:

- *Movie attributes*: These are unique for each movies and outside of control of distributors. They include theme acceptability, viewer satisfaction, perceived movie quality… etc. The price is not included as it tends be constant across movies.
- *Marketing strategy*: how the movie is advertised through the media or word-of-mouth.
- *Distribution strategy*: It includes the release date, the initial number of theaters…etc.
- *Adoption structure*: It consists of the average time needed for consumers to forget the movie is playing, and average time to spread active word-of-mouth messages (Positive or Negative).

The movie release strategy plays a key role on the box office revenues. Three distinct release strategies are identifies:

- *Wide-release*: Large advertising campaign and large number of theaters for the initial screening.
- *Platform-release strategy*: Local advertising, small number of initial theaters. It increases within two to four weeks after the release.
- *Limited-release strategy*: few or no advertising, very small number of theater for the initial screening with no expectations of wider release.

The model uses rule-based logic to assign the most appropriate release strategy to a movie in the database, but this is outside the scope of this thesis.

The dataset used is constructed from movies released in 2009 containing for each movie: weekly box-office receipts theater counts and descriptive variables.

More information on the model itself are available at C. Hughes [57] and for more information regarding dynamic modeling J. Sterman [10].

6.1.2 Current issues

This model is slow to run. The main reason behind this is that every time the model is called it reload the whole database. It results in a 9 seconds run on our configuration. Unfortunately this source code of the model was not available to us because it is designed with Anylogic [58] and its license is expensive. The designer of the model will update it in the future.

Moreover, it is worth mentioning that this model is still in development. These are the only known issues at the moment.

6.2 Role of the Particle Swarm Optimization

6.2.1 Connection with the swarm

In order to make accurate predictions, the model needs a set of parameters. At this stage, the goal of the model is to make predictions for released movies. In this way, the model can compare its output to the historical box-office revenue and then compute the squared error (computed within the model itself). The role of Particle Swarm Optimization here, is to fine tune the parameters of the model for a given movie in

order to maximize the inverse squared error. We did not have control over the fitness function and had to use the one provided. (Further collaboration will improve this aspect, we believe).

The first challenge was to be able to run programmatically the model which was run by a script through the command line. The initial idea was quite complex, uses depreciated JAVA library, and failed several times. The solution was to use the ProcessBuilder class provide by the JAVA API. It gives the possibility to call external program within the JAVA code. The final code is simple:

```
ProcessBuilder pb = new ProcessBuilder("cmd.exe", drive, file);
pb.directory(new File(path));
Process p = pb.start();
```

We simply needed to specify which program to call, “cmd.exe” with Windows, the location and the script through the variables “drive” and “file”. At line 2 we specify the working directory. And finally run the command at line 3.

The PSO required to load parameters in the movie, select the movie and retrieve its output, all programmatically. Fortunately, the model uses separated text file for each of these requirements:

- “Input_param.txt”: one parameters per line, very easy to read and write.
- “output_box.txt”: simple double value to compute the fitness.
- “input_readMovie.txt”: just the index of the movie to be analyzed.

To write into a text file we used the New Input Output library of JAVA:

```
ArrayList<String> tmp = new ArrayList<>();
tmp.add(st);
Files.write(path, tmp, ENCODING);
```

The method “Files.write” takes only a list of String as parameter, this is why an ArrayList is declared at line 1. At line 2 we add to the list the desired String to write, in our case, either a list of parameters or the index of a movie. Finally, line 3 writes the string into the file at the location “path” with the proper encoding (we used UTF-8)

We implemented the following structure to be able to communicate back and forth between the PSO and the model:

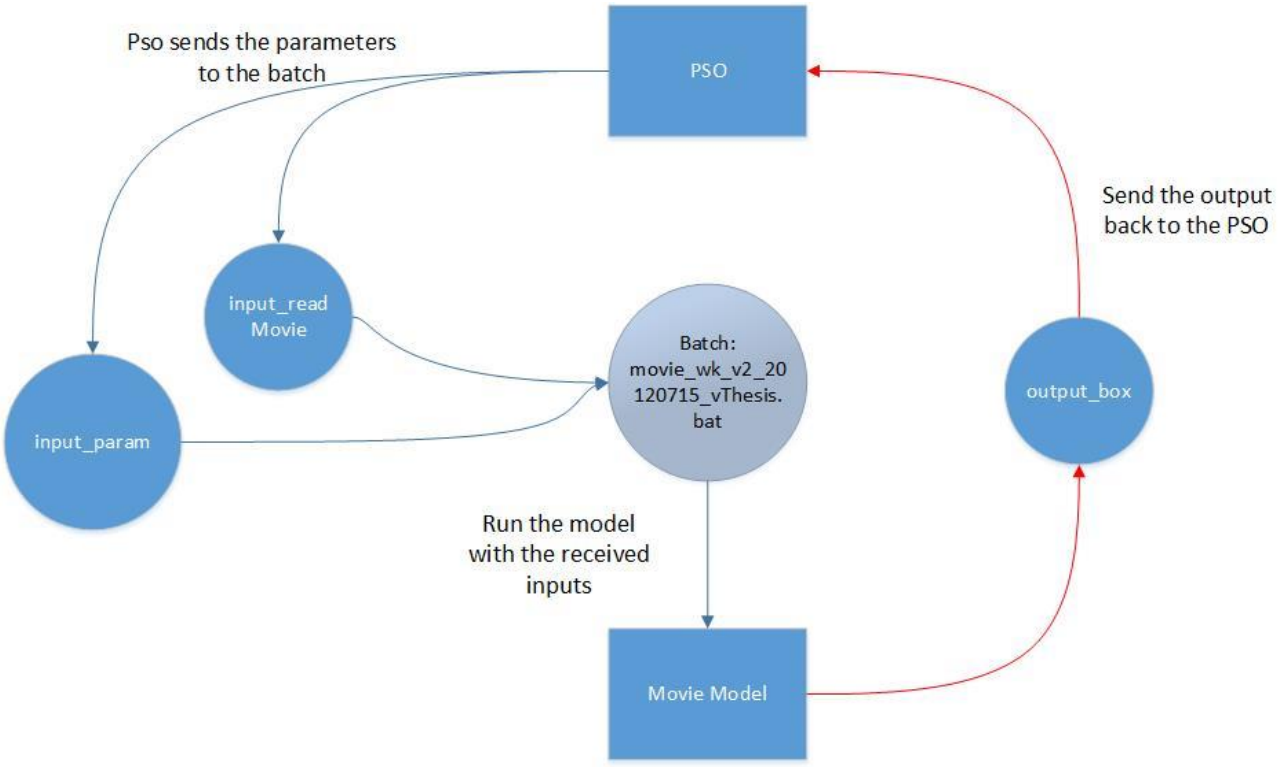


Figure 6.3: The Movie model interaction

The PSO sends the potential solution of a particle in the "input_param.txt" file and specify the movie in the same. Then run the model, which sends back its output in the PSO. This design worked fine, but create multiples dependencies of the PSO on the model, which is usually considered as a bad design.

Our collaborator Mr. Hughes suggested to extend this experiment to any kind of dynamic model. However, this design was not suited for such a task. Indeed, the PSO depends of the model's structure. We applied a proxy pattern to fix this problem.

The proxy pattern is a class functioning as an interface to something else, usually a complex object that is expensive to call or duplicate. In our case, we created a proxy for the movie model, once all the parameter for the movie model are set in the proxy. The proxy forward them to the model, allowing the model to be called only when it is absolutely necessary.

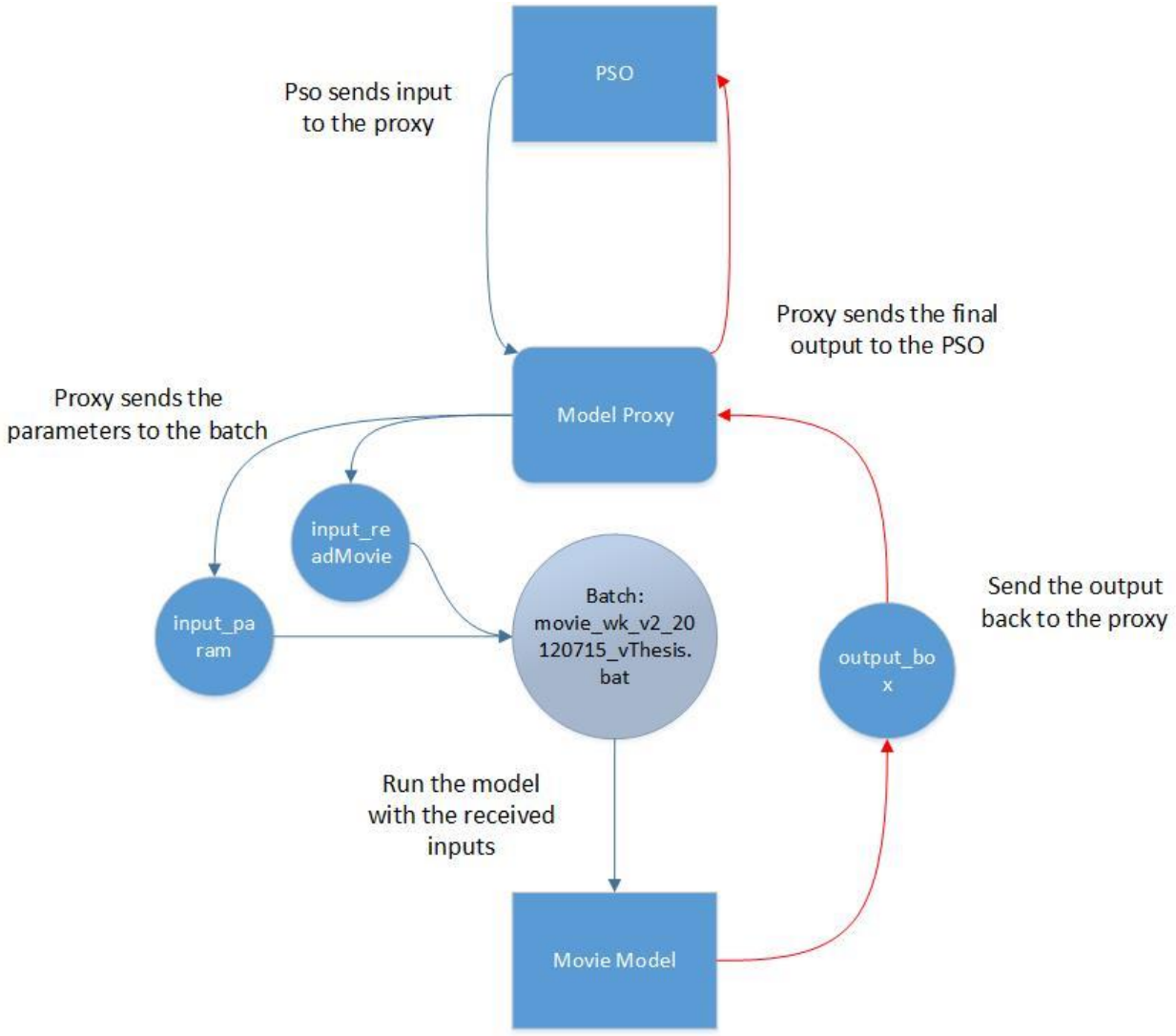


Figure 6.4: Movie model/PSO interactions with proxy

This pattern completely breaks the dependency of PSO toward the model. It only relies on the proxy's implementation to run, allowing us to use any kind of dynamic model with minimum modification, only the proxy's parameters need to be modified.

6.2.2 Fitness function

The position of a particle represents a set of feasible parameters of the model. The fitness of such parameters is evaluated by comparing the performance of the model compared to the historical performance of the movie being studied. This concept is illustrated in Figure 6.5. The evaluation of the model is done within the model, and we did not have control over it. This is the reason why the fitness function evolves between negative infinity and one. We noticed that the model perfectly matches the historical performances of the movie.

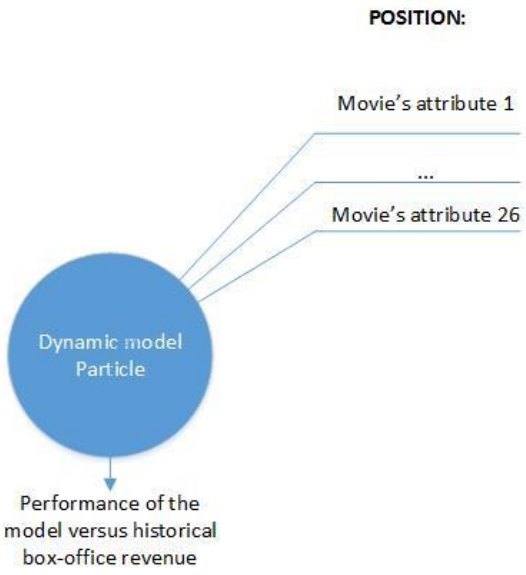


Figure 6.5: Dynamic particle model

6.3 Experiments

Due to the slow running time of the model, around thirty minutes per movie, we selected only a few movie to optimize. Mr. Hughes in PWC recommended certain movies that are challenging due to complex release strategies. Furthermore, each movie is unique and we cannot compare the final fitness between movies. Hence, we want to compare the fitness obtain with the PSO with the fitness obtained by a previously used method (classical optimization method). Table 6.1 compares the results of the PSO against the initial performance of the model. As we can see, PSO outperforms the classical approach on 8 movies out of 11. The experiments were performed several time. We observed that the PSO is sometimes worst or equivalent to the classical approach as for the movie "Transformers: Revenge of the Fallen" (index 636). This is due to the random initialization of the swarm and the stochastic element in the velocity equation. Indeed, the random starting point of the swarm might initialize the swarm, far from an optimal solution. Hence, the convergence is slower when we stopped the optimization after 10 iterations, the swarm ended up in a position close to the classical method or worst. Furthermore, the stochastic element might not help to improve the performance. Its role is to encourage the exploration of the search space at the cost of driving some particles away from a near-optimal solution.

The following Table 6.1, compares the result of the swarm Optimization with the classical approach. We selected 11 movies out of the database. As we can see, the PSO outperformed the classical method on 8 movies with, sometimes, very large improvement in the performance of the model (cf: "Sunshine Cleaning", index 595).

Title (database index)	Fitness (PSO)	Fitness (classical method)
Sunshine Cleaning (595)	0.542041	-0.01124
Terminator Salvation: The Future Begins (623)	0.973589315	0.765042455
Away we go (629)	0.46346917	0.246040903
Transformers: Revenge of the Fallen (636)	0.841458566	0.898601731
The hurt locker (637)	0.661136538	0.455146655
500 Days of Summer (644)	-0.170461313	0.399418732
Taking Woodstock (663)	0.848825872	0.540585803
9 (669)	-1.090255369	0.08923669
Capitalism: A Love Story (677)	0.730379031	0.540698928
Paranormal Activity (681)	0.75274435	0.487752198
An education (686)	0.507373612	0.397807509

Table 6.1: Movie model results

However, the non-linear nature of the model and its highly dynamical propriety (each movies possess a unique set of parameters) along with the slow running time of the model showed that a more advanced collaboration with the PWC company is required. This experiment holds as a proof of concept for the next development step of this project. Nevertheless, these results confirmed that this approach had a great potential for such problems.

Chapter 7 - Study Case: DNA classification

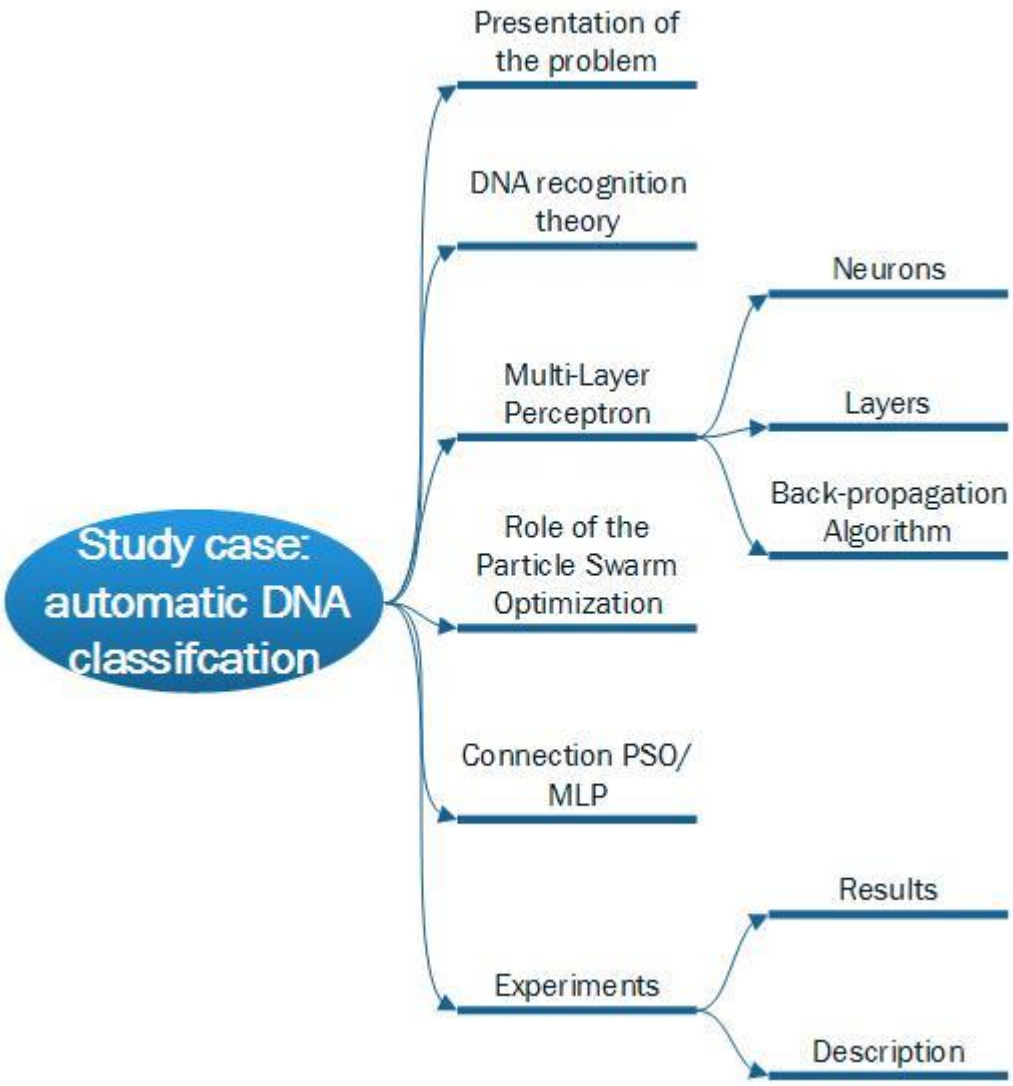


Figure 7.1: Mind map Chapter 7

7.1 Presentation of the problem

One of the main research goals of molecular biology has been to determine a complete genetic description of any organism. In the Human Genome Project [59].the goal was to decipher the exact sequence of about 3 billion nucleotides in the 46 human chromosomes. An important part of the genome project is the computational processing of data [60]. The data first have to be organized into databases, and then analyzed to see what information they contain. Since the birth of the Human Genome Project, sequence analysis as a computational method has been used to infer biological information from the sequence data.

The classical approach for analyzing sequences is by sequence matching using either single or multiple alignment techniques [61] [62]. With these techniques one seeks to determine whether sequences are significantly similar or not. Another approach is to use theories from neural computing or statistical learning theory to detect genetic information on the DNA sequences.

Neural networks have been applied to various tasks such as automatic hyphenation of natural languages [63] [64], edge detection [65], recognition of hand written Zip code and DNA sequence recognition [66]. A neural network trained by a Back propagation algorithm (BP) may learn to categorize between different types of bacteria cells related to the structure of their DNA-sequences. Such a method is based on pattern recognition analysis, and is built on the assumption that some underlying characteristics of the DNA-sequence can be used to identify its bacteria type. Other neural network paradigms than a MLP network may also be used to analyze DNA sequences [67].

In this paper, however, we will focus on how to use both a Multi-Layer Perceptron (MLP) and a Support Vector Machine (SVM) network to distinguish between eukaryotic and prokaryrotic sequences on basis of their nucleotide frequency structure. Cells can be divided into two major groups, prokaryotic and eukaryotic cells. All prokaryotic cells are uni-cellular organisms and consist mostly of bacteria. The genome of a prokaryotic cell consists of one double helix DNA strand, floating freely in the cell. This double helix strand is often circular. The genome of the bacterium E.Coli, for instance, consists of a circular strand of five and a half million bases.

A nucleotide sequence can be viewed as a language based on an alphabet of four letters: A, G, C and T where the number of A's is the same as the number of T's and the number of C's is the same as the number of G's. However, the relation of A(T) and G(C) can vary tremendously, and depends on the actual species that are studied. This fact can for instance be used in environmental research, where oil on the sea surface may contain many different types of species that can be identified on basis of their DNA sequence structure.

Most eukaryotes are multicellular, but some are uni-cellular. The main difference between prokaryotic cells and eukaryotic cells is that eukaryotic cells contain a nucleus that is surrounded by a membrane. Prokaryotic cells do not have such a nucleus. In such cells the frequency distribution of pairs of nucleotides are different from those in prokaryotic cells.

We will train a MLP network and a Support Vector Machine using PSO and compare the result with our previous research paper [68]. But first, we need to introduce the DNA recognition theory.

7.2 DNA recognition theory

Statistical analysis of several DNA sequences has shown that the distribution of nucleotides is far from random [17]. Some dinucleotide combinations in prokaryotic DNA sequences are more dominating than in eukaryotic cells. We will anticipate that this simple difference in data occurrence might be sufficient to allow species identification. We may then train, for instance a MLP network (see section 7.2.4), to use the differences in the nucleotide distribution to discriminate between eukaryotic and prokaryotic cells. We assume therefore that the identification of the DNA sequence is based solely on the frequency of nucleotide sub-sequences.
A sliding window is used to count the number of nucleotide sub-sequences of the DNA sequence. In general the size of the window may vary, from one base wide to a user defined number *w*. By choosing a window of length one, we simply count the number of the different bases of the DNA sequence. The result will be four different frequencies, one for each base. A window of length two will give sixteen different ordered sub-sequences. The frequency of each sub-sequence is computed by counting the occurrence of each nucleotide pair in the DNA sequence.

The number of triplets or *codon* units of the DNA sequence, may be estimated by using a window of three bases wide. This results in 4^3 or 64 ordered triplets. This is maybe the most relevant sub-sequence to study because the codon itself has important meanings in the DNA sequence. In general, a window of *w* bases results in 4^w sub-sequences of length w to be counted.

For a sliding window of length two the frequency of sub-sequence AA is denoted as f_{AA}, for AC as f_{AC} and so on. These numbers are collected in a vector $\mathbf{F_n}$, where n denotes the number of DNA sub-sequences. For a sliding window of size two, n is equal to 16. The counting of the different nucleotide pairs is illustrated in Figure 7.2. In the figure the counting of the nucleotide pair AC is shown. After counting the pair AC, the window is moved one letter to the right to cover the next nucleotide pair. This is done to the end of the DNA sequence.

ACATGATGCTA...

ACATGATGCTA...

ACATGATGCTA...

ACATGATGCTA...

Figure 7.2: Illustration of a sliding window of size 2

Normalization:

The DNA sequences obtained on online databases have different sequence length. The frequency of the different nucleotide pairs have to be normalized before and presenting them to the MLP network. The normalization condition of the frequency vector F_n is given by:

$$S_n = \frac{F_n}{|F_n|}$$ *Equation 7.1*

Here $|F_n|$ means the Euclidean norm of the frequency vector. This non-linear transformation conserves the direction of the vector and enhances the differences among the input vectors. The geometric interpretation of the transformation is that the vector F_n is moved onto the hyper unit sphere.

In our case, we will use a sliding window of size three and incorporate the redundancy of the genetic code (see Appendix A). It means that the representation of the genetic code will be built into the neural network by use of the frequency vector.

7.2.1 Multi-Layer Perceptron

In this chapter we will study the application of PSO on a supervised learning algorithm: A multi-layer Perceptron. ANN has been known for a long time and is inspired by the biological neural network. In the brain, it is composed of artificial neurons connected by synapses and organized by layers.

The MLP has three basic features:

- Each unit (neuron) includes a differentiable nonlinear activation function.
- The network contains one or more hidden layers.
- The network is highly connected: each unit of a layer is fully connected to every unit of the next layer.

These features makes the theoretical analysis of the network difficult for two reasons. Firstly, the high connectivity and the nonlinearity of the network makes the analysis hard to undertake. Secondly, the presence of hidden layer makes the learning process difficult to visualize.

We are going to use the back propagation (BP) algorithm to train the network. The training requires a set of data points associated with their corresponding target vector. It means, we have data that we know to which cluster they belong, allowing us to use them to train the network.

The BP algorithm consists of two phases:

- *Feed forward phase*: the synaptic weight are fixed and the input signal is propagated through the network layer by layer and neuron by neuron, from the input layer toward the output layer.
- *Back propagation phase*: The network computes an error signal by comparing the output of the network to the desired output (target vector). This error is propagated backward, layer by layer from the output layer toward the input layer. In this phase the weights are adjusted according to the error signal.

This two phases are applied until the error reaches a threshold value defined by the user.

MLP uses two types of signals:

- *Function signals*: they are defined as input signals at the input of network, propagates forward neuron by neuron to emerge as output signal at the end of the network.
- *Error signals*: They are defined by output neurons using an error-dependent function. It propagates backward layer by layer through the network.

7.2.2 Neurons

An artificial neuron is an entity that receives one or several signals as input. Each input signals arrive through a synapse, it simply assigns a weight to the original signal representing the strength of the signal. Each unit preprocess the signals by computing the following summation that is called the induced local field:

$$y = \sum_{i=0}^{n} x_i w_i$$ *Equation 7.2*

Then a sigmoid function (or logistic function) is applied to compute the output signal of the current unit:

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$ *Equation 7.3*

The input neuron (which receives the value from the data points being evaluated) is the only one that does not compute anything. It just transmit the input signal through its synapses. X_0 is a bias applied to every unit. The bias is nothing more than the multiplication of an input vector with a matrix. Using a bias adds another dimension to the input space, which always takes the value one to avoid input vector of zeros.

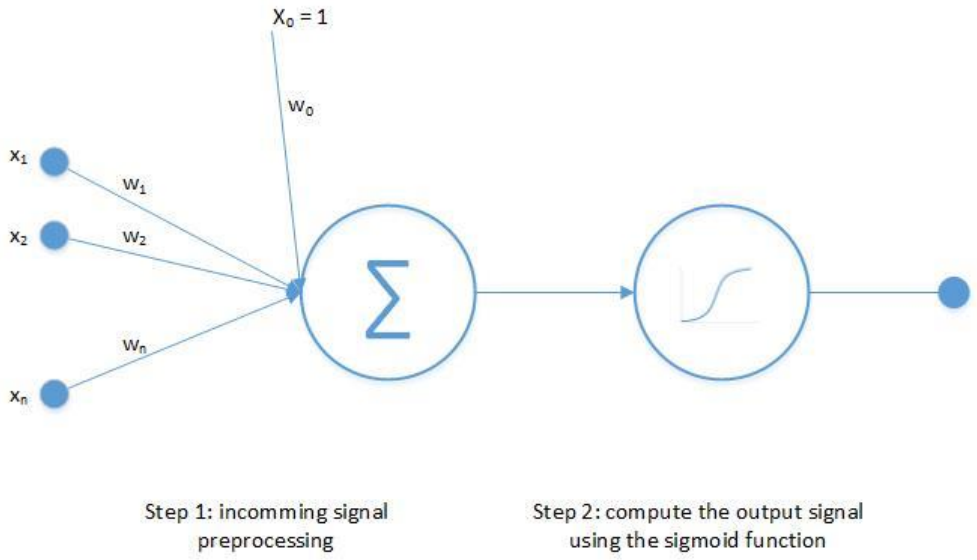


Figure 7.3: Sigmoid unit.

A single neuron cannot classify anything. However, neurons are organized in layers as we can see in the next section.

7.2.3 Layers

A MLP is composed of three type of layers:

- An input layer: it transmits the input data point through its synapses to the others layers. It does not compute anything.
- One or several hidden layers: it receives the signals coming from the input layer and computes output signals using the sigmoid function and finally sends its signals to the output layer.

- An output layer: it contains as many unit as there are clusters (or classes) in the dataset. Each unit computes an output signal as in a hidden layer.

Figure 7.4 shows a typical architecture of a Multi-layer Perceptron with one hidden layer.

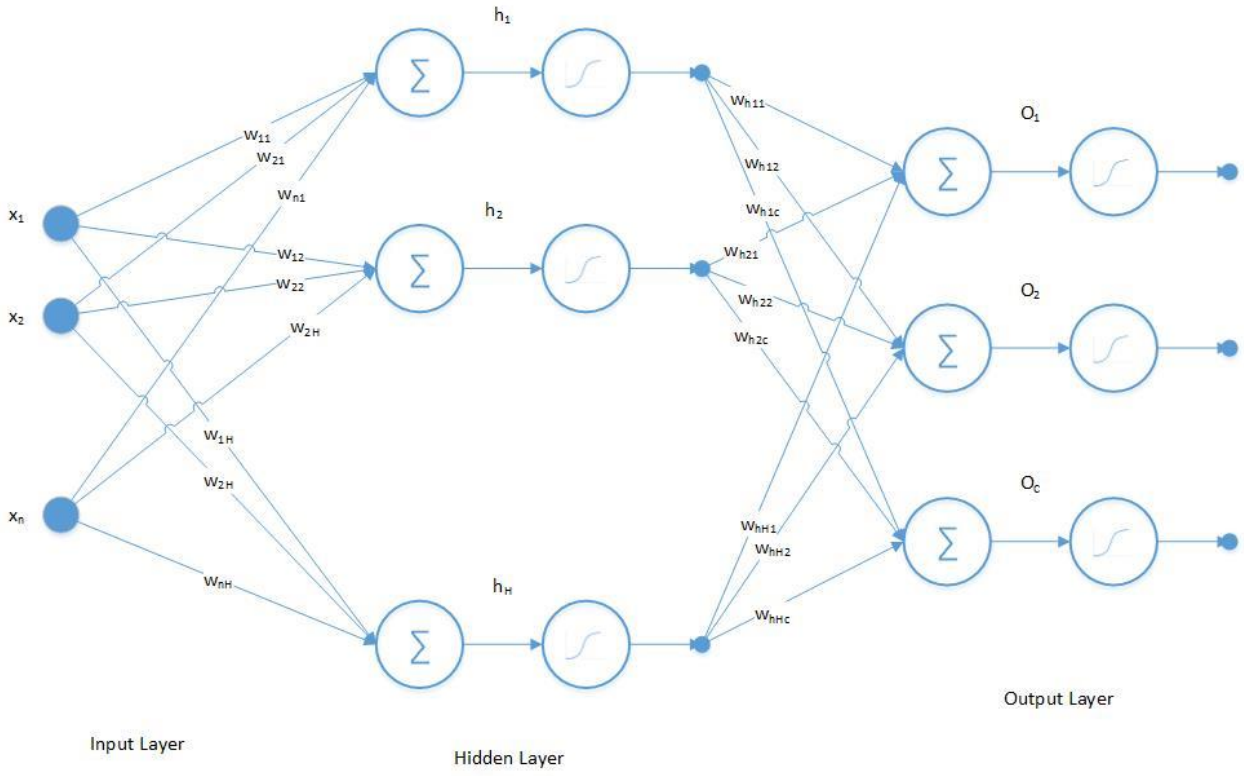


Figure 7.4: The MLP architecture

The structure of a MLP is given in Figure 7.4.

7.2.4 Back-Propagation Algorithm

For a classification problem of dimension n, with c clusters, we create L-2 hidden layer. Then the network has L layers (also referred as the depth of the network).

The BP algorithm consists of five steps:

1. Initialization: the weight matrix is initialized using a uniform distribution.
2. All the training examples are presented once, the network perform one forward and one backward propagation (respectively step 3 and 4).
3. Forward propagation: we denote a training example $\{x(t), d(t)\}$, $x(t)$ is presented as input vector to the input layer of the network, and $d(t)$ is applied to the output layer as a target vector. The signal of each unit is computed layer by layer using the induced local field equation. Then the sigmoid function is applied to obtain the *function signal.* The induced local field $v_j^l(t)$ for training example t, neuron j and layer l is:

$$v_j^l(t) = \sum_i w_{ji}^{(l)}(t)\, y_j^{l-1}(t) \qquad \textit{Equation 7.4}$$

Where $y_j^{l-1}(t)$ is the output signal of unit j of layer l-1 and $w_{ji}^{(l)}(t)$ is the weight of synapse between unit j in layer l and unit i in layer l-1 at iteration t. Assuming a sigmoid function we have:

$$y_j^l(t) = \begin{cases} x_j(t) & \text{if l=0, i.e. input layer} \\ o_j(t) & \text{if l=L, i.e. output layer} \\ \dfrac{1}{1+e^{-v_j^l(t)}} \end{cases} \qquad \textit{Equation 7.5}$$

Once every induced local field and function signal are computed, the error signal is evaluated using the following equation:

$$e_j(t) = d_j(t) - o_j(t) \qquad \textit{Equation 7.6}$$

Where d_j is the j[th] element of the desired output vector $d(t)$ and o_j is the j[th] element of the real output vector $o(t)$ obtained from the output layer. Once the error vector is obtained, the forward propagation ends.

4. Backward propagation: it uses the error vector previously computed to compute the local gradient. Indeed, the back propagation tries to find the steepest vector in the error function in order to minimize the error for the next iteration. The local gradient δ is defined by:

$$\delta_j^{(l)}(t) = \begin{cases} e_j^{(L)}(t)\varphi_j'\left(v_j^{(L)}(t)\right) & \text{for neuron j in output layer L} \\ \varphi_j'\left(v_j^{(L)}(t)\right)\sum_k \delta_k^{(l+1)}(t) w_{kj}^{(l+1)}(t) & \text{for neuron j in hidden layer l} \end{cases} \qquad \textit{Equation 7.7}$$

where φ' is the differentiation of the sigmoid function with respect to the argument. And k is varying through the connected neurons of neuron j.

Finally the synaptic weights in layer l are adjusted using the equation:

$$w_{ji}^{(l)}(t+1) = w_{ji}^{(l)}(t) + \alpha\left[w_{ji}^{(l)}(t-1)\right] + \eta\delta_j^{(l)}(t) y_i^{(l-1)}(t) \qquad \textit{Equation 7.8}$$

where α is the learning rate and η is the momentum.

5. Iteration: forward and backward propagation are repeated until a chosen stopping criterion is met (often the error reaching a predefined threshold).

Once the BP algorithm terminates, the weights are fixed and the network can be used to classify unseen data points. A major difficulty with MLP is to find the right value for the different parameters of the network: number of hidden units, number of hidden layer, learning rate and momentum. Another issue is the time consuming training. If the learning performs too well, the network can be over trained. It means that the network perform very well on the training data, but will fail to classify unseen data points.

For a complete proof of the BP algorithm, see to Mitchell [69].

7.3 Role of the Particle Swarm Optimization

The BP algorithm tries to minimize an error function. This could be done by using a PSO. This is a common approach, instead of using the BP algorithm, swarm optimization can be used to find the weight matrix that minimize the error function. However, we will use the PSO to find the other parameter of the network. Indeed, a particle will represent a possible configuration of the network. In other words, the position of a particle will be the following vector: (learning-rate, momentum, number of hidden neuron).

The weight matrix is determined using the back-propagation algorithm.

7.3.1 Structure

In this study case we try to use PSO to determine the parameters of the MLP and the SVM networks.

The support vector Machine uses a Gaussian Kernel (see section 2.5). This means we need to determine two parameters: gamma and the cost (Figure 7.5 A). The MLP needs 3 parameters to be determined: the learning rate, the momentum and the number of hidden unit (the input and output layer are already determined) (Figure 7.5 B).

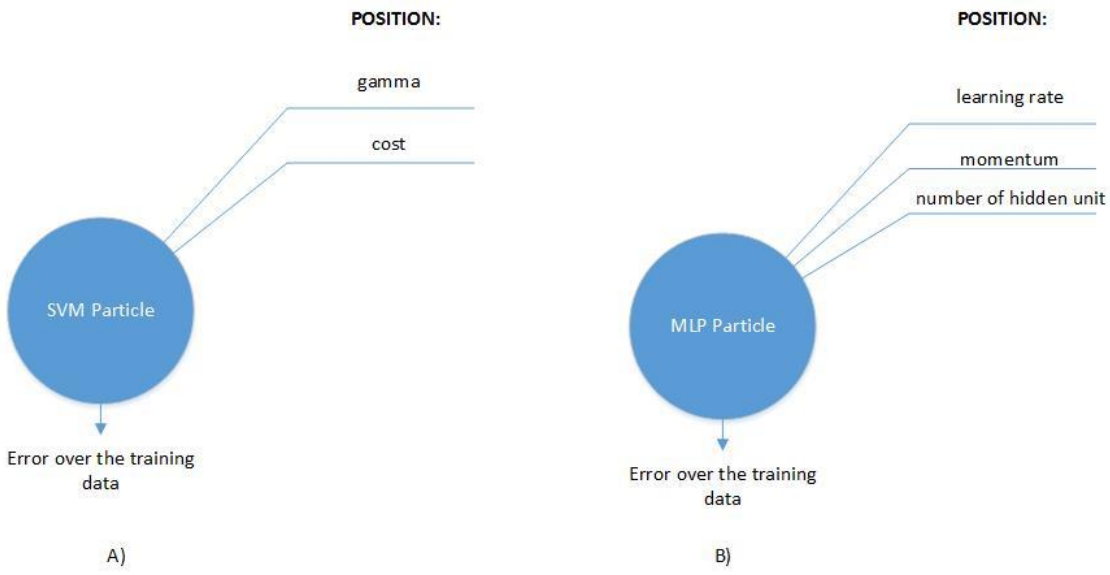


Figure 7.5: A SVM/MLP particle

The implementation of the MLP was done using the Neuroph library [70]. While the SVM was implemented by using LIBSVM [71]

The program works in several steps:

1. The data is stored in text files containing the DNA sequences. The 'IOManager' class calls the 'TextReader' class to extract the DNA sequences from our text files, and converts them into 'Sequence' class.

2. Now that the sequences are loaded, the 'IOManager' computes the frequency vector of each sequence using the 'Alphabet' class, which contains the representation of the codons of the genetic code to identify triplet of amino acids as codon.
3. Now the fitness of either the MLP or the SVM network is initialized, followed by the generation of the swarm.
4. The PSO is executed and the data retrieved at the end.

The following diagram shows the structure of this study case. As reminder the class 'PSOexperiment' is a class that defines a template to create an experiment to run on the PSO, while the 'Fitness' interface defines an abstract fitness function to make the PSO functional.

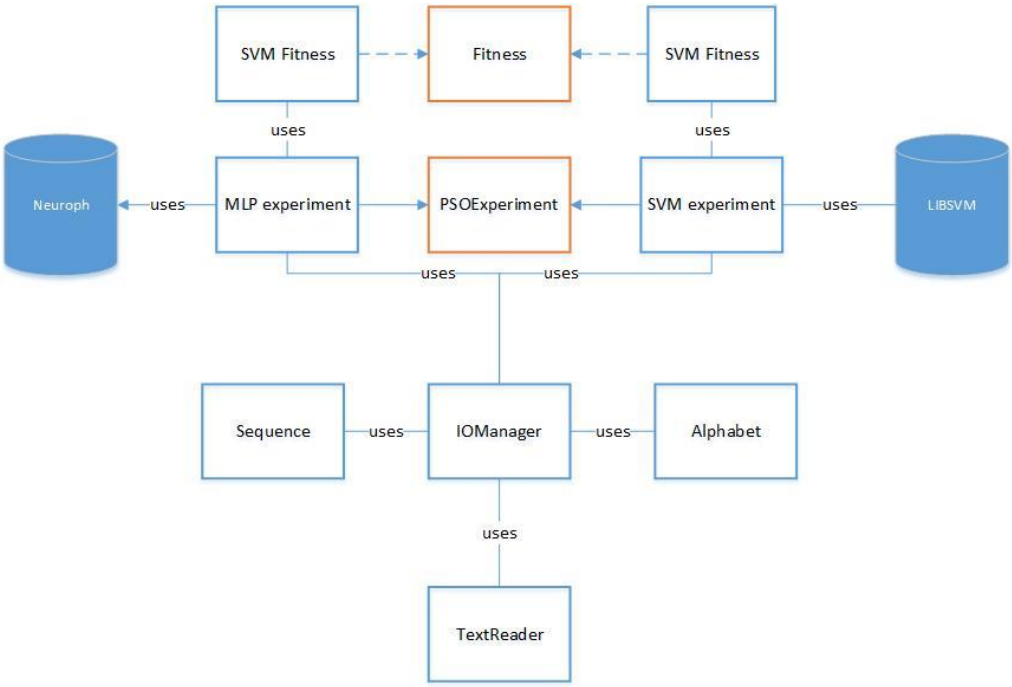


Figure 7.6: A MLP-SVM class diagram

7.4 Experiment

7.4.1 Description

The dataset we used was composed of 54 sequences: 28 eukaryotic sequences and 26 prokaryotic sequences. Figure 7.7 shows the type of file we have to deal with. The process required to automate the extraction of the DNA sequence within the file (the two last line in the illustration). All files were not identical but every DNA sequences were tagged with "SQ", the extraction was a straight forward String analysis of the file. The common approach is to split the data set in two, one part to train the SVM or the MLP, another to test their performance. However, in our case the data set is too small to be split, the MLP does not perform well with too few training data. We decided to evaluate each classifiers on the training data only. In order to limit the training time of the MLP we limited the number of iteration to 10000. The swarm was composed of 20 particles separated in two sub swarms. And we used a fixed number of

iterations: 10. In total, the MLP has been trained 2000 times like the SVM. This gives an idea how computationally expensive such optimization can be. It all depends on the fitness function, as we saw in the movie model.

7.4.2 Results

The PSO performed well to find the configuration of the neural network. The parameters obtained are the following:

Learning rate	Momentum	Hidden neurons	**Error**	Running time (min)
0.83	0.3	59	0.0096	11.5

Table 7.1: Final MLP results

While the configuration of the SVM is:

Gamma	Cost	**Error**	Running time (sec)
0.58	975	0.074	3.6

Table 7.2: Final SVM results

Our first conclusion are the following, as we can see the SVM does not perform so well. The data showed that the custom-PSO algorithm was not able to properly optimize the SVM classifier, it converged after only two iterations and didn't improve afterwards.

To confirm this results, we reproduced the experiments using the global PSO (Algorithm 3.1). The swarm had the following parameters:

- Initial size: 25 particles
- Inertia coefficient: 0.95
- Cognitive coefficient: 0.9
- Social coefficient: 0.9
- Number of iterations: 10

The results for the MLP are the following:

Learning rate	Momentum	Hidden neurons	**Error**	Running time (min)
0.78	0.82	75	0.0094	17

And for the SVM:

Gamma	Cost	**Error**	Running time (sec)
0.8	624	0.074	5

The results are similar. Regarding the MLP, we observe a high learning rate in both case. And large hidden layer, the second experiments gave a high momentum too. And both experiments reached a low error. However, the downside is a long training time. Regarding the SVM, in the second experiments, the gamma was higher with a lower cost, but the error was identical, and the running time still low.

By doing these experiments, it confirmed the ability of PSO to optimize an MLP and an SVM. However, it also showed the limit of the SVM in this particular DNA classification problem. The MLP outperformed the SVM but had a very long training time.

However, explaining and exploring this issue would require a work far beyond this thesis. We simply make experiments with the algorithm and make observations regarding its capacity to resolve certain problems.

```
ID   AD2VAIP    standard; DNA; VRL; 62 BP.
XX
AC   K00523;
XX
SV   K00523.1
XX
NI   g210018
XX
DT   02-JUL-1986 (Rel. 09, Created)
DT   16-MAR-1992 (Rel. 31, Last updated, Version 3)
XX
DE   adenovirus 2 vai rna gene promoter region.
XX
KW   promoter.
XX
OS   Human adenovirus type 2
OC   Viruses; dsDNA viruses, no RNA stage; Adenoviridae; Mastadenovirus.
XX
RN   [1]
RP   1-62
RX   MEDLINE; 84093106.
RA   Bhat R.A., Metz B., Thimmappaya B.;
RT   "organization of the noncontiguous promoter components of adenovirus vai
RT   rna gene is strikingly similar to that of eucaryotic trna genes";
RL   Mol. Cell. Biol. 3:1996-2005(1983).
XX
CC   the internal promoter regions were determined by mutants containing
CC   deletions, insertions and substitutions. the distance between the a
CC   and b regulatory sequences must be longer than 35 base pairs for
CC   transcription to take place.
XX
FH   Key             Location/Qualifiers
FH
FT   source          1. .62
FT                   /organism="Human adenovirus type 2"
FT                   /db_xref="taxon:10515"
XX
SQ   Sequence 62 BP; 12 A; 15 C; 22 G; 13 T; 0 other;
     TCCGTGGTCT GGTGGATAAA TTCGCAAGGG TATCATGGCG GACGACCGGG GTTCGAACCC        60
     CG                                                                      62
//
```

Figure 7.7: A DNA sequence file

Chapter 8 - Study Case: Support Vector Clustering

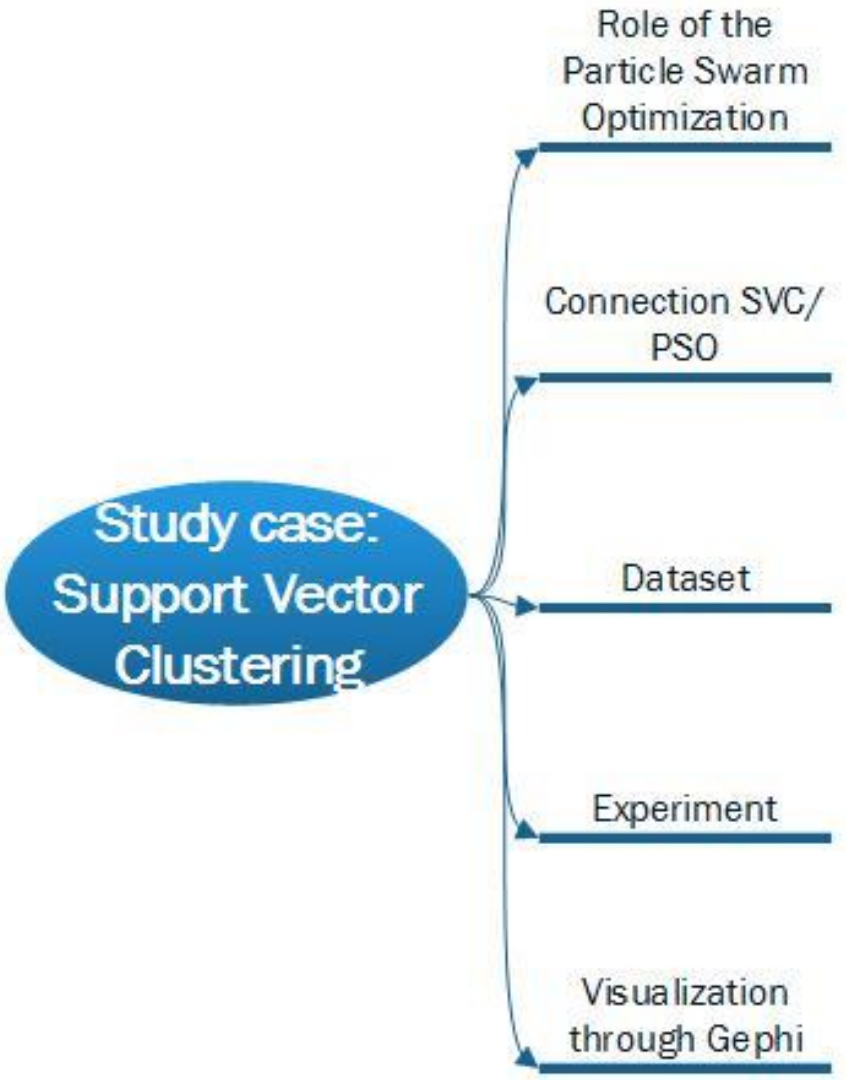


Figure 8.1: Mind map Chapter 8

8.1 Role of the Particle Swarm Optimization

Swarm Intelligence has been used to find the parameters of a Support Vector Machine [72] [73] [74]. We are going to applied PSO to determinate near-optimal Eigenvector for the Support Vector Clustering.

We presented the Support Vector Clustering in section 2.6. As we saw this is a quadratic optimization problem to solve. The classical approach involves numerical analysis methods or more advanced algorithms such as the Sequential Minimal Optimization algorithm (SMO) [75]. The SMO has been developed only for support vector machine but not clustering. A version for SVC exists but we did not find any information about it. Numerical analysis can be heavy to run and to understand. While SMO is very fast, the algorithm is hard to comprehend. We are going to use Particle Swarm Optimization to solve this optimization problem. The PSO will try to maximize the Lagrangian (Equation 2.34). In other words, it will try to compute the Eigenvector. Once the PSO has terminated, the SVC resumes its clustering process as explained in section 2.6. The following flow chart illustrates the new clustering procedure:

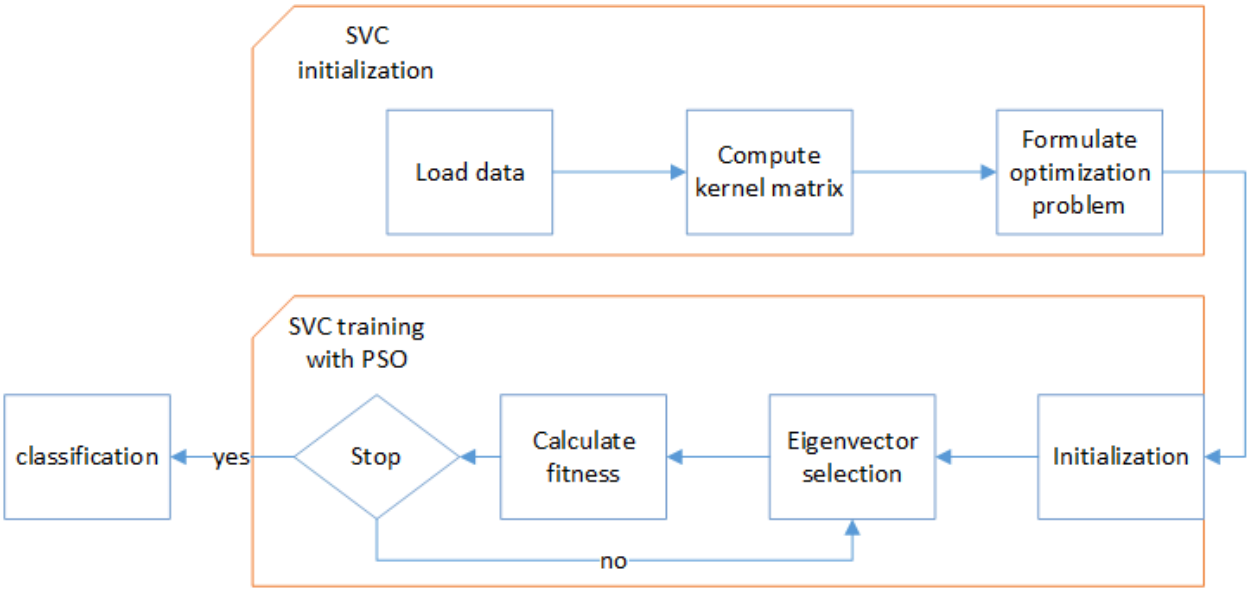


Figure 8.2: A SVC/PSO procedure

As we can see, the kernel matrix is computed once and for all, as it does not depend on the Eigenvector. By computing it only once, we avoid to do this expensive computation every time we evaluate the position of a particle. The PSO output its solution to the SVC, which resume its clustering procedure.

The classification consists of the following steps:

1. The PSO find a near-optimal Eigenvector
2. The distance of every data points from the center of the hyper-sphere in the feature space is computed by using Equation 2.39.
3. Support Vectors, Bounded Support Vectors are identified.
4. The radius of the hyper-sphere is simply the distance of a Support Vector from the center of the hyper-sphere.
5. The adjacency matrix is computed as follow:

a. For all pair of data point, we select every points on the line defined by a pair of points. We test the co-linearity of vectors i.e. for every pair (A,B) a point C is on the line (AB) if and only if $\overrightarrow{AB}$ and $\overrightarrow{AC}$ are co-linear:

$$\forall i, j, \quad AB_i AC_j = AB_j AC_i \qquad \textit{Equation 8.1}$$

b. Then we check if for all point C on the line (AB), we have:

$$R(C) \leq RADIUS \qquad \textit{Equation 8.2}$$

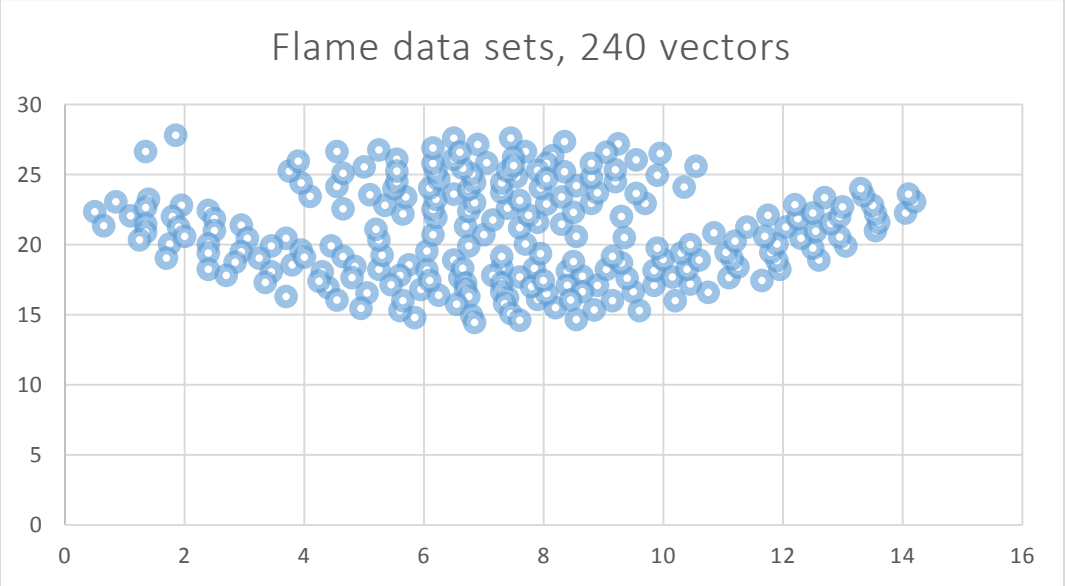
c. Steps ***a*** and ***b*** consists of Equation 2.42

8.2 Connection SVC/PSO

The connection between the two algorithms is made in a similar fashion as for the movie model. Using a controller, the SVC calls directly the PSO to find a near-optimal Eigen-value. Then the adjacency matrix is computed and exported in the software Gephi using the Gephi API [50].

8.3 Dataset

We used a data set called "Flame" [76] of 2 dimensions, composed of 240 vectors containing two clusters. The following scatter plot represents the data set, no clusters are labelled as no analysis has been run yet on these data points. The scatter is just the actual data points on a two-dimensional plot. In other words, this is the raw data.



The two clusters are clearly visible.

8.4 Experiment

We tried different values for the cost and gamma. However, the output was always the same. The cost was fixed at 1 and the gamma value was set at 2. After several tests, we set up the swarm with 20 particles and 100 iterations. The same evolution of the swarm occurred at every experiments. Figure 8.3 is an example of this pattern. The swarm always starts to drop in the fitness landscape and tends to stay in a

low fitness region for about half of the time of the optimization process. Then, the whole swarm increases drastically its performances, but slowly decreases afterwards.

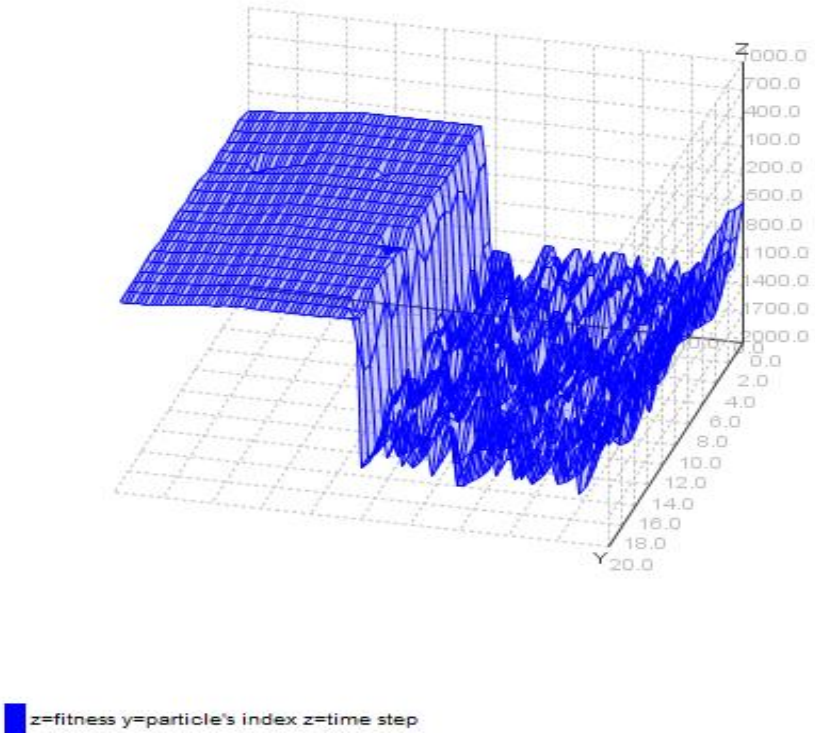


Figure 8.3: A PSO 3D plot of SVC

The following figure shows the behavior of one particle through the optimization process. We can see more clearly this pattern.

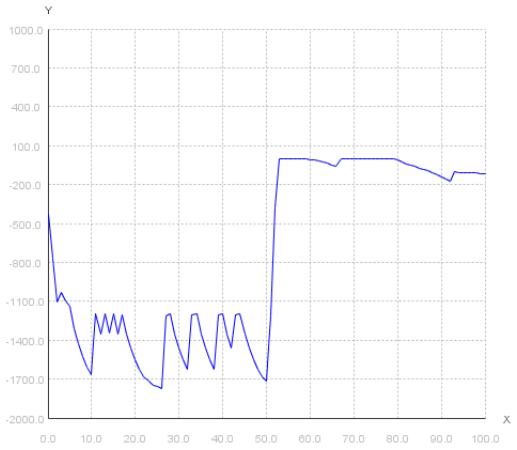


Figure 8.4: The evolution of the fitness of a particle SVC.

Several trials have been done. However, increasing the length of the optimization process did not changed the outcome. The swarm still has the same behavior. We used different behaviors of the particle such as the cognitive-only model and the social only-model, but the fitness achieved was the same. The adjacency matrix computed from the PSO's solution was a nearly fully connected graph. Such a graph has only one strongly connected component leading to the discovery of only one cluster by the algorithm.

Unfortunately, it means that the PSO is unable to find the Eigenvector for the Support Vector Clustering algorithm because the SVC procedure requires the exact solution of its QP equation, and the PSO provides only approximations.

8.5 Visualization through Gephi

Nevertheless, we proceeded with the visualization of the adjacency matrix. An undirected graph is created from the previously computed adjacency matrix. The graph is composed of 240 nodes and around 28 600 edges. Once the graph is created, it is loaded in the open source software Gephi [50]. The ForceAtlas2 [77] algorithm is applied to the graph. This algorithm re-arrange the nodes of a graph to regroup highly connected nodes. It improves the readability of a graph and highlights strongly connected component, which correspond to the clusters discovered by the SVC in our case. Figure 8.5 shows the result of the visualization process. As mentioned previously, the PSO was unable to find the proper Eigen-vector. The direct consequences of this, is the failure of the SVC to discover any cluster in the data.

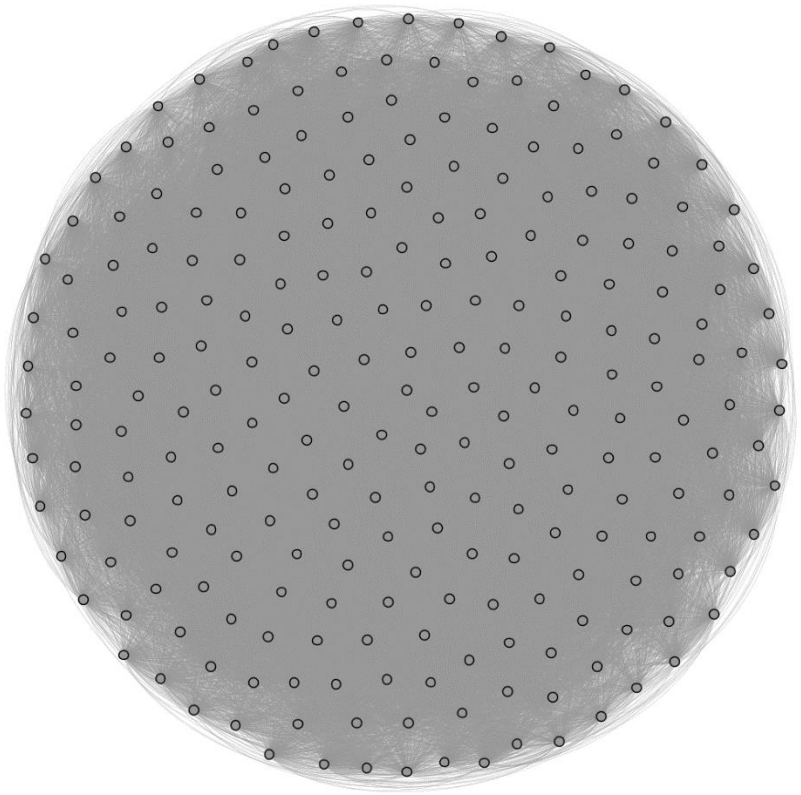


Figure 8.5: Gephi visualization

This last experience shows a limit of our custom PSO algorithm. Not as a poor quadratic optimizer, but as a poor solution for the Support Vector Clustering. This experience demonstrates that SVC does require an optimal solution, “near-optimal” is simply not good enough for such a task.

Chapter 9 – Conclusion and further work

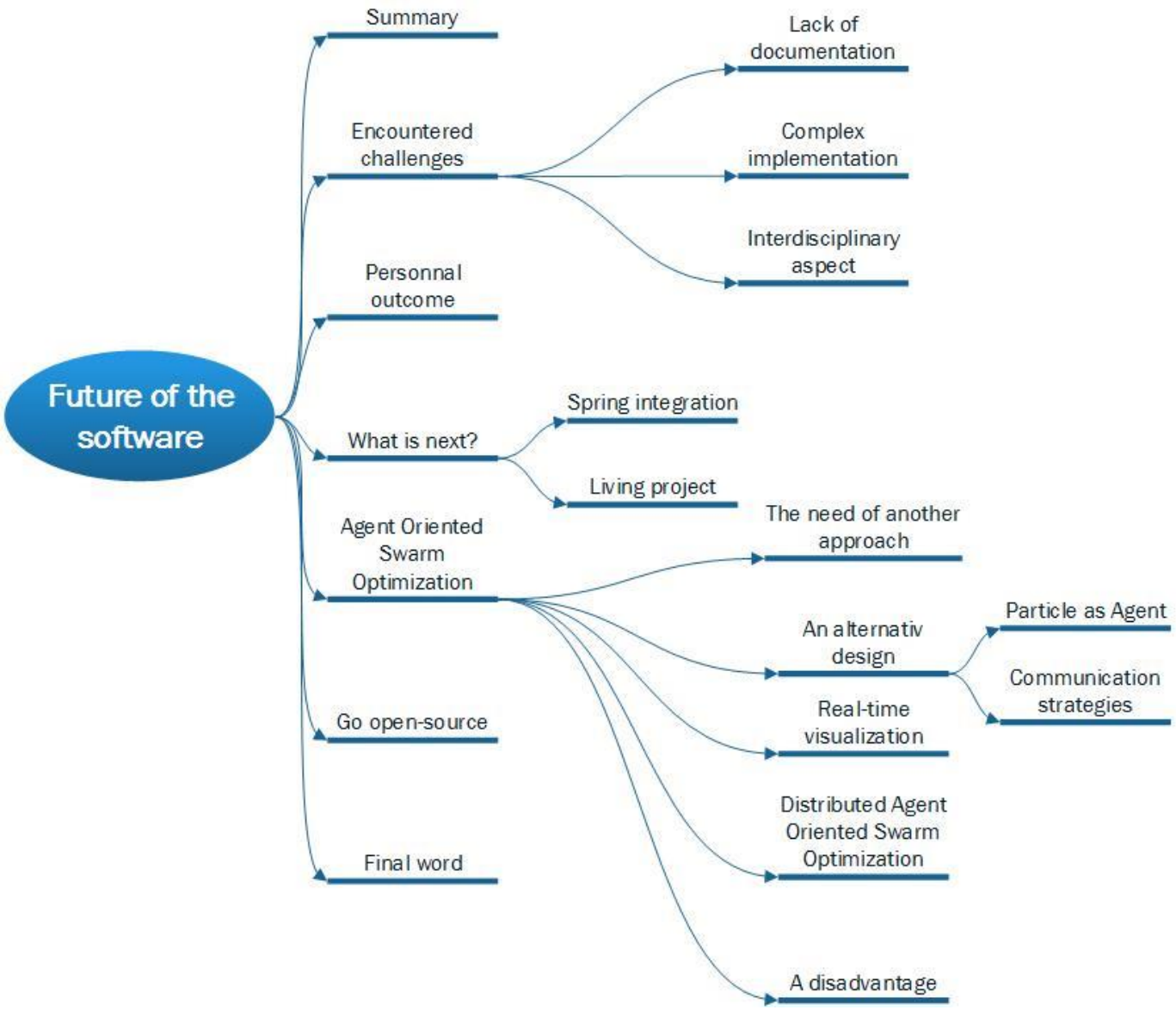


Figure 9.1: Mind map Chapter 9

9.1 Summary

During the first experiment, PSO demonstrated a great ability to explore the parameter space of the movie model. It achieved large fitness values with few iterations. The second study case uses PSO to find the parameters of a Multi-Layer Perceptron and a Support Vector Machine in order to classify two types of DNA sequences. The PSO gave a noticeable improvement in the performance of the MLP, but a less noticeable increase in performance of the SVM. However, the two first study cases demonstrate a great ability of PSO to explore parameter spaces. And we can consider extending this application to the classification more than two types of DNA sequences.

Unfortunately, the last study case using Swarm Optimization to find the Eigenvector of the Support Vector Clustering algorithm has failed. Indeed, it appears that a near-optimal solution for the SVC is not good enough to compute with accuracy the radius of the hyper-sphere in the feature space. This inability leads to a completely wrong adjacency matrix because of the approximate solution is not optimal. In the end, the SVC was unable to discover any clusters in our dataset, even though the data set contained only two clusters with a clear separation. PSO is able to find parameter where exact solutions are not an absolute requirement. However, for SVC, it seems that mathematical exactness is necessary, and a near-optimal Eigenvector is not enough.

9.2 Encountered challenges

We summarize the main challenges encountered within this research project.

9.2.1 Lack of documentation

The Support Vector Machine is very well documented, a lot of implementation exists, even custom optimization algorithm have been created to find the Eigenvector [77]. However, Support Vector Clustering is close to SVM in the form (it is kernel based and involves Quadratic Programming) but it is completely different since it is unsupervised learning. We came across one main article about it [78]. The main principle was clear but very few details on how to solve the Eigenvector. A version of the Sequential Minimal Optimization algorithm [77] has been made for SVC, no implementations nor articles about it were available. Furthermore, the method to compute the adjacency matrix was unclear as again no detailed explanation was given.

9.2.2 Complex implementation

We made the choice of a Swing based JAVA application right from the beginning as it seemed to be the most efficient way. Unfortunately, the size of the program turned bigger and more complex than expected. This type of application is not suited for such a task. As the decoupling between the different logical parts (PSO, SVC and the study cases) and the visualization part was particularly difficult to achieve. We believe it has been successfully done after the architecture of the program was changed for the third time. We realized this problem too late for the project to finish before the deadline.

9.2.3 Interdisciplinary aspect

This thesis belongs to the field of Computational Intelligence and Software Engineering. By nature, it is deeply interdisciplinary as PSO, for instance, it is inspired by the work of biologist, since the SVC algorithm

is highly mathematical, mixing both of them were very interesting, and maybe applied to a wide variety of problems

9.3 Personal outcome

In order to complete such a project, we needed to learn rigorous planning. We followed a plan with daily objectives and deadlines to meet. It gave a great insight on project management, and how hard it can be to meet deadlines.

While doing research about the different technologies we could use, we had to try many programming languages (JAVA, JavaScript, Dart, C++). Before selecting a specific library, we have to learn how to use it, discover their strengths and weaknesses. In a few word, the technical outcome was tremendous.

9.4 Next steps

9.4.1 Living project

This project is far from being finished. Indeed, our cooperation with Mr. Hughes at PwC to optimize the movie model revealed two observations:

5. The need of efficient optimization software for such a model, as the current optimization techniques are not so efficient. The field of dynamic modeling is relatively new, and comes from the business world. Today not so many computer scientists have been worked in this field.
6. Working with Mr. Hughes at PWC and presenting the PSO algorithm to his coworkers exhibited another point. It is difficult to introduce an optimization techniques such as PSO. In other word, the challenge was to explain that “heavy” mathematics is not required to create powerful optimization algorithms. Effective communication with no software engineering people makes it difficult to explain how our approach could lead to new perspective in the modelling process.

From these points, we decided to push the project further in this direction after the thesis ends. The goal is to develop an optimization algorithm for a dynamic model. It has to be easy to use and we want that the user select only a few options prior the run of the optimization process. The options are:

- The algorithm (global PSO, local PSO, etc…). However, the different parameter values of each algorithm will not be available to the user.
- And the choice between two swarm behavior: *exploration*, *exploitation*. Sometimes in dynamic modelling, we are not looking for one unique solution. We want to explore the search space as much as possible to detect interesting features (sub optimal regions, for instance). And *exploitation* to try to find the best solution as quick as possible (with a higher risk of finding only local optimal).
- An alternative view, more technical, will give full control of the parameter of the algorithms to the user, like: the size of the swarm, termination conditions, behavior of the particles etc…
- And finally, this program must be independent of the operating system. The JAVA programming language is well suited for such a requirement as it needs only the Java Virtual Machine to be run. However, we will use a more advanced solution presented in the next section.

This engineering process makes a challenge to the modeler too, as we will need to create templates for the models in order to load them in the optimization program. A first version of this template is already created. However, it needs further refinement to be easier to use.

9.4.2 Spring integration

As mentioned earlier (in 9.2.2) the limitation of a Swing based application makes the evolution of the program difficult. Even if JAVA is portable, we decided to transform the current program in an enterprise application based on the Spring framework [79]. This framework uses the Java Enterprise Edition model, and makes the implementation of the MVC easier. Indeed, it is based on tiered enterprise application

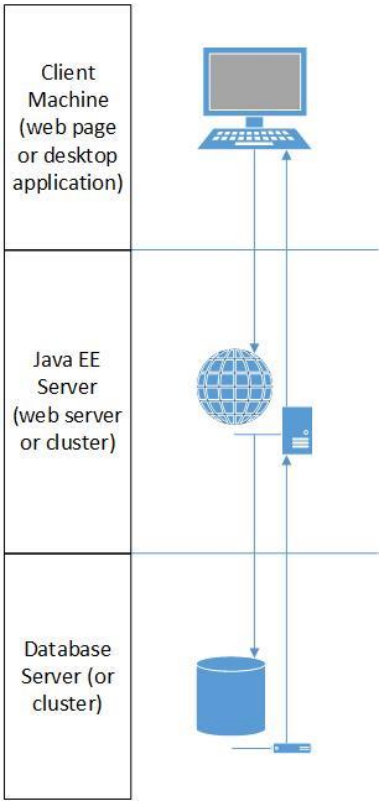


Figure 9.2: Enterprise Application

Figure 9.2 shows the concept of enterprise application. The middle tier execute all the logic. In our case, it will execute the PSO. The database will store the different models to be optimized. And lastly, the client will use a web page to set up its experience. This point is fundamental to improve the user experience. We will be able to use all the power of HTML5/CSS3 and Javascript to create our user interface, and visualization algorithm, in a very user friendly way. The user will have nothing to install, he will just need to continue to the corresponding web page, load its model, and run the optimization algorithm. Furthermore, the client side is completely disconnected from the logic. It will give us a greater freedom to change the algorithm or even implement a new one.

9.5 Agent Oriented Swarm Optimization

9.5.1 The need of another approach

At the moment, the Swarm implementation is hierarchical as shown in Figure 9.3. This means that when the PSO algorithm execute the move command on the swarm, it is first passed to the swarm, then to the sub swarms and finally reaches the particles. It works in the same way for any other commands as fitness evaluation or finding the best particle. Furthermore, every evaluation of the fitness function is made on the same Java Virtual Machine. This solution works just fine for our study cases. It would be useful to run simultaneously with several fitness evaluations on many JVM, or even many processors. In other words, develop a parallel version of PSO. We also would like to reach an implementation closer to the biological model for two reasons:

- A new architecture could improve the performance
- More flexible implementation could help to distribute (on a cluster of servers for instance) the computation for very intensive optimization problems.

The only way to know is to implement such architecture and test it.

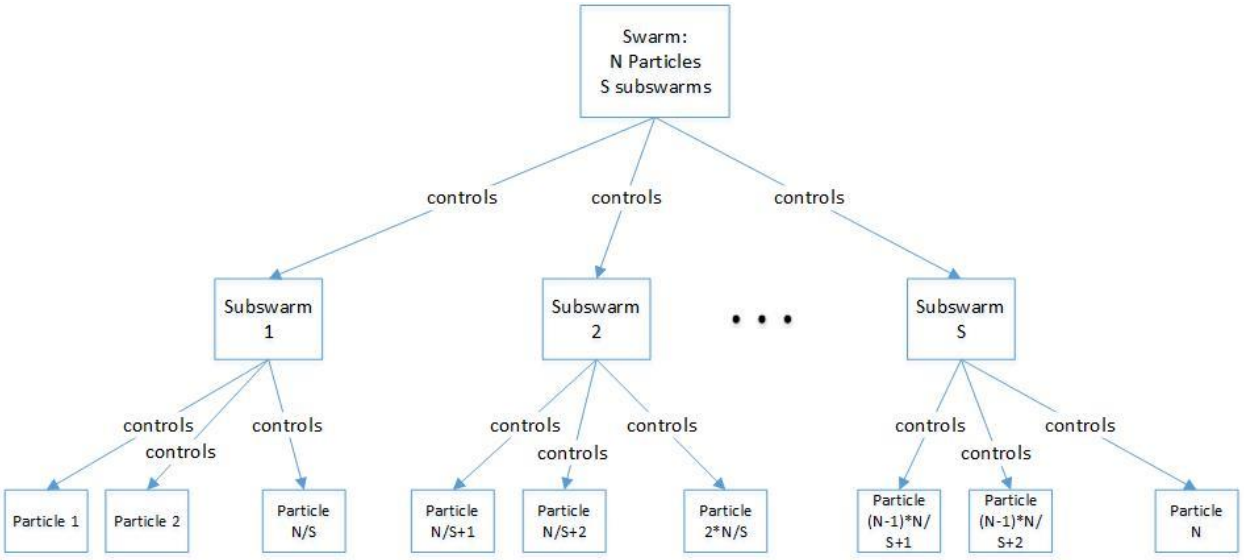


Figure 9.3: Hierarchical swarm

9.5.2 An alternative design

9.5.2.1 Particle as Agent

Our new architecture is based on Multi-Agent System (MAS) [80]. We are going to use the following definition of an agent:

> *An agent is a computer system that is situated in some environment, that is capable of autonomous action in this environment in order to meet its delegated objectives.*

This definition matches the role of a particle. From now on, we will explain MAS only through Particle Swarm Optimization to avoid unnecessary explanations. Three notions are important in this definitions. Firstly, an agent lives in an environment, the search space of our fitness function. Secondly, the agent is capable of autonomous action, it can move in the environment, and communicate with other particles (request for the best particle for instance). Finally, an agent has delegated objectives, and it has to find the optimal possible value of a fitness function, based on [Russel and Norvig, 1995, p32] [81]. A particle as agent can be viewed as follows:

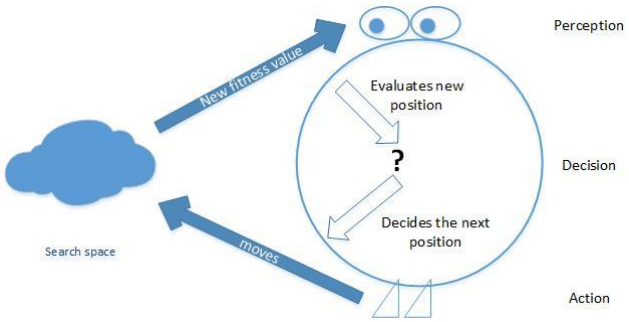


Figure 9.4: A Particle-Agent

An agent is an entity well suited for a particle.

9.5.2.2 Communication strategies

Agents can be social. It means that they can communicate with each other using a messaging system. This will be useful to find the best solution discovered by the swarm or by a subswarm. Let's assume a swarm of N Agents, and a request for the best known solution. The MAS uses an interface agent (IAg) to handle all the user request.

We may have two strategies to handle such request:

- IAg sends a request for the best know solution to every agents, he processes all the messages once every agent has answered. This approach sends a lot of messages, and could lead to a too high usage of the network.
- IAg sends one request to the first particle (P1), the message contains the request, and the list of the particle who already received this request, only P1 at the beginning. P1 then creates an answer with his best known solution, and add his name to the list within the message. Then, it forwards this message to the next particle that is not in the list. The following particle compares its best known solution to then one in the message. If the new solution is better, it is specified in the message. The particle adds its name in the list, and forward the message to another particle. The message jumps from particle to particle until every particle has received the message. At this point, the best particle has been found (the message has been through the whole swarm looking for the best position). The last particle send the message back to the IAg. This strategy uses less messages than the previous strategy. With this agent-based approach, only four messages are exchanged since six messages are sent in the first method.

Figure 9.5 illustrates both approaches.

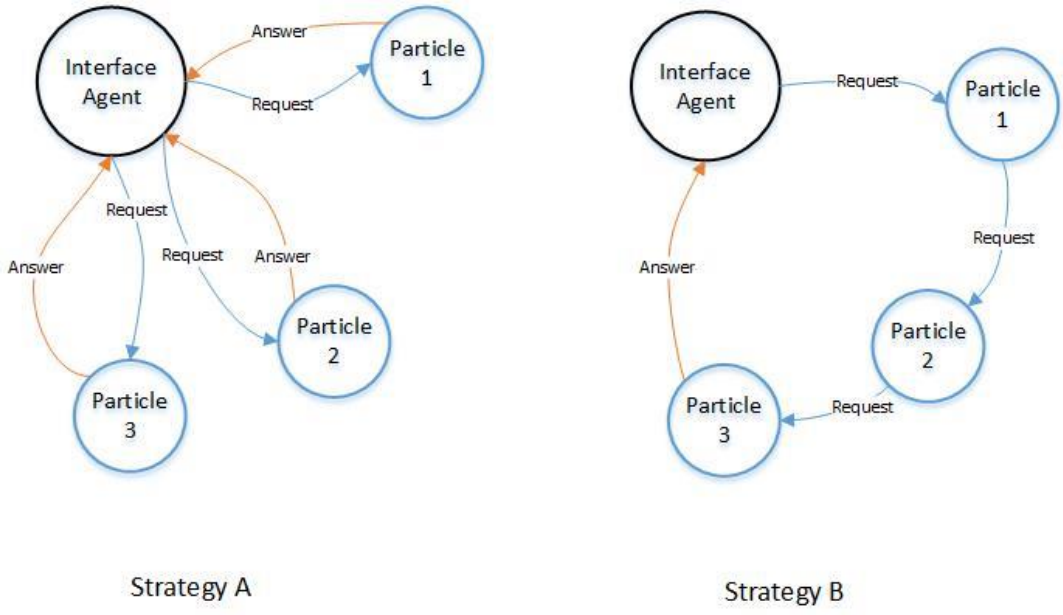


Figure 9.5: A Communication strategy

We want to implement the second strategy as it uses less messages. Furthermore, it does not require to sort the particles to find the best solution. It is both easier to implement and faster to execute.

9.5.3 Real-time visualization

Each agent will use a system of event listener. Every time a particles moves, or evaluates its position, it will trigger an event. This event will send back the data of the agent in a database and alert the interface agents that something has changed. The interface agent will then render the new state of the swarm by retrieving information in the database. This system will allow us to visualize the swarm without interfering with it, at any moment. Figure 9.6 shows the flow of messages when the particle moves.

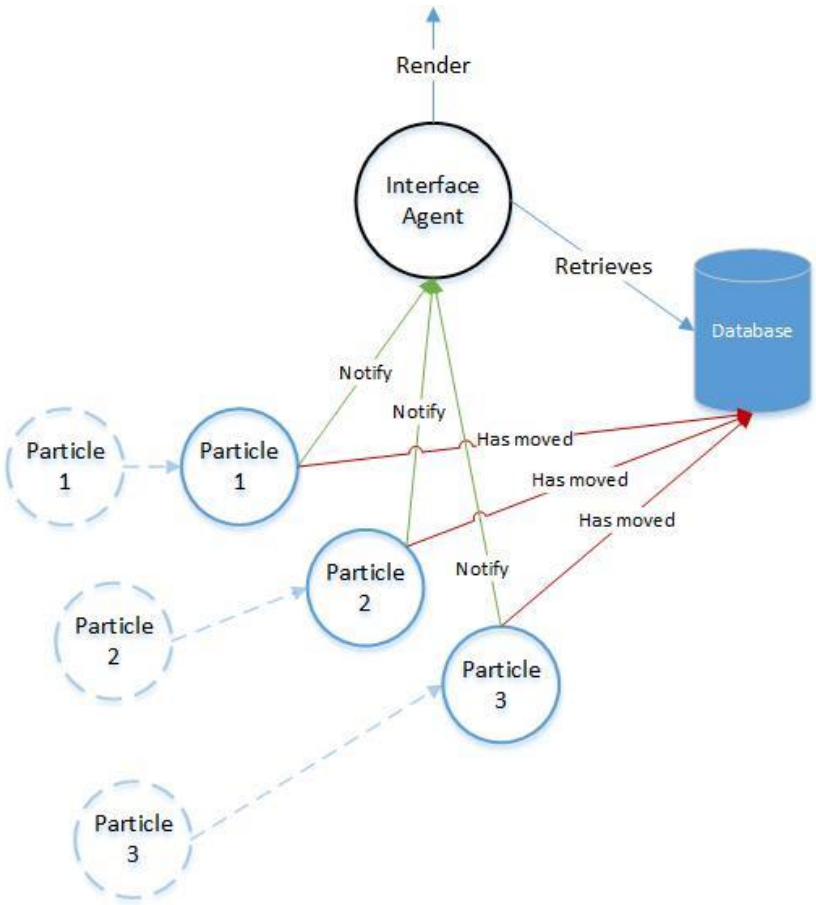


Figure 9.6: A Particle's movement.

9.5.4 Distributed Agent Oriented Swarm Optimization

It is important to remember that as Agents, the particles do not need to reside on the machine. This feature is particularly interesting when it comes to heavy optimization problems. We can create the swarm in a cluster of servers. In this way each particle can move from one server to another depending on available resources. We call these particles, *migrating particles. They are* more complex to be implemented, but they will be facilitated by the use of JADE [82]. JADE is probably the most advanced platform to create and maintain a Multi-Agent System. It facilitates the migration of agent between servers, and contains its own message system and is written in JAVA. These three points may make JADE the right platform to use in the future project.

9.5.5 A disadvantage

Such a new architecture has one disadvantage. The particles become independent. It is then difficult to update the particles in a synchronous way, i.e., the notion of iteration becomes unclear, as particles can easily be updated one by one in any order. However, we choose to operate the swarm in a asynchronous way. Every particles will move freely with a delay between each move. However, they are not going to wait until every particle in the swarm move one time step before it starts a new displacement. This approach gives more freedom to the particle.

9.6 Final word

Finally, the software developed will move into the Open-source community. It seems to be the right choice as it can gather more people on the project. More people means more opinions, and it can only stimulate the development. This is not going to happen yet, as a website and forum needs to be created.

This thesis lead us to new exciting challenges. However, we would like to explore the different aspects of PSO and clustering in the future.

Today, three distinct new projects might emerge from our work:

- The first one, was presented in Section 9.4 and 9.5 and it is an agent based PSO library.
- The second one, is an independent optimization software tailored for dynamical model.
- And finally, we would like to create a platform implementing various clustering algorithms. The goal behind these ideas is the same. We want to make unsupervised learning and Swarm Optimization more accessible to a wider number of researchers and not only to experts in datamining and optimization.

Finally, the thesis was an incredible learning experience.

References

[1] T. Hofmann, A. J. Smola and B. Schölkopf, "Kernel Methods in Machine Learning," *The Annals of Statistics,* vol. 36, 2008.

[2] K. McKinnon, Convergence of the Nelder-Mead simplex method to a non-stationary point, vol. 9, Siam J Optimization, 1996, pp. 148-158.

[3] A. P. Engelbrecht, Computational Intelligence: an introduction, Wiley, 2007.

[4] T. Kristensen, Neural Networks, Fuzzy logic and Genetic Algorithms, Cappelen Academic Publisher, 1997.

[5] G. Guojun, M. Chaoqun and W. Jianhong, Data Clustering: Theory, Algorithms and Applications, Siam, 2007.

[6] T. M. Cover, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, Vols. EC-14, IEEE Transactions on Electronic Computers, 1965, pp. 326-334.

[7] K. Y. Yeung, C. Fraley, A. Murua, A. E. Raftery and W. L. Ruzzo, Model-based clustering and data transformations for gene expression data, Bioinformatics, 2001.

[8] W. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis.," vol. 24, no. 5, pp. 603-619, 2002.

[9] M. Christopher, "Cluster analysis and market segmentation," vol. 3, no. 2, pp. 99-102, 1969.

[10] J. Sterman, Business Dynamics, Cambridge: Irwan McGraw-Hill, 2000.

[11] B. Everitt, S. Landau and M. Leese, Cluster analysis, 4 ed., London: Arnold, 2001.

[12] X. Rui and W. I. C. Donald, Clustering, Wiley, 2009.

[13] L. Zadeh, "Fuzzy sets," Information and control, 1965, pp. 338-353.

[14] R. Duda, P. Hart and D. Stork, Pattern classification, New York: John Wiley & Sons, 2001.

[15] R. Hogg and E. Tanis, Probability and Statistical Inference, Upper Saddle River: Prentice Hall, 2005.

[16] B. Everitt, Cluster analysis, 2 ed., London: Social Science Research Council, 1980.

[17] S. Haykin, Neural Network and Learning Machine, Hamilton: Pearson International Edition, 2009.

[18] S. Boyd and L. Vandenberghe, Convex Optimization, New York: Cambridge University Press, 2004.

[19] B. Schölkopf and A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond, MIT Press, 2002.

[20] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations.," *Transaction of the London Philosophical Society,* pp. 415-446, 1909.

[21] R. Bellman, Dynamic Programming, Princeton: Princeton University Press, 1957.

[22] J. Shawe-Taylor and N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge and New York: Cambridge University Press, 2004.

[23] D. M. Tax and R. P. Duin, "Support vector domain description," *Pattern Recognition,* pp. 1991-1999, 1999.

[24] R. Fletcher, Practical Methods of Optimization, Chichester: Wiley-Interscience, 1987.

[25] R. J. Vanderbei, Linear Programming Foundations and extensios, 3 ed., Princeton: Springer, 2008.

[26] J. Kennedy and R. Eberhart, Particle Swarm Optimization, Proceedings of IEEE International Conference on Neural Networks, 1995.

[27] J. Kennedy and R. Eberhard, A Discrete Binary Version of the Particle Swarm Algorithm In Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, 1997, pp. 4104-4109.

[28] R. Eberhard, P. Simpson and R. Dobbins, Computational Intelligence PC Tools, Academic Press Professional, 1996.

[29] J. Salerno, Using Particle Swarm Optimization Technique to Train a Recurrent Neural Model, Proceeding of the IEEE International Conference on Tools with Artificial Intelligence, 1997.

[30] Y. Shi and R. Eberhart, Empirical Study of Particle Swarm Optimization, Proceedings of the IEEE Congress on Evolutionary Computation, 1999.

[31] Y. Shi and R. Eberhart, A Modified Particle Swarm Optimizer, Proceeding of the IEEE Congress on Evolutionary Computation, 1998.

[32] F. van den Bergh, "An Analysis of Particle Swarm Optimizers, PhD thesis," Pretoria, 2002.

[33] M. Clerc and J. Kennedy, The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space, IEEE Transactions on Evolutionary Computation, 2002.

[34] J. Kennedy, The Particle Swarm: Social Adaptation of Knowledge, Proceedings of the IEEE International Conference on Evolutionary Computation, 1997.

[35] A. Carlisle and G. Dozier, Adapting Particle Swarm Optimization to Dynamic Environments, Proceedings of the International Conference on Artificial Intelligence, 2000.

[36] K. Parsopoulos, D. Tasoulis and M. Vrahatis, Multiobjective Optimization using Parallel Vector Evaluated Particle Swarm Optimization, Proceedings of the IASTED International Conference on Artificial Intelligence and Applications, 2004.

[37] R. Brits, "Niching Strategies for Particle Swarm Optimization, Master Thesis," Pretoria, 2002.

[38] R. Brits, A. Engelbrecht and F. van den Bergh, "A Niching Particle Swarm Optimizer," *Proceedings of the Fourth Asia-Pacific Conerence on Simulated Evolution and Learning,* pp. 692-696, 2002.

[39] M. Clerc, "Think Locally, Act Locally: The Way of Life of Cheap-PSO, An Adaptive PSO," 2001.

[40] J. H. Holland, Adaptation in Natural and Artificial Systems, Ann Harbor: University of Michigan Press, 1975.

[41] C. Koay and D. Srinivasan, "Particle Swarm Optimization-Based Approach for Generator Maintenance Scheduling," *Proceedings of the IEEE Swarm Intelligence Symposium,* pp. 167-173, 2003.

[42] H. Higashi and H. Iba, "Particle Swarm Optimization with gaussian Mutation," *Proceedings of the IEEE Swarm Intelligence Symposium,* pp. 72-79, 2003.

[43] C. Wei, Z. He, Y. Zheng and W. Pi, "Swarm Direction Embedded in Fast Evolutionary Programming," *Proceedings of the IEEE Congress on Evolutionary Computation,* vol. 2, pp. 1278-1283, 2002.

[44] B. Secrest and G. Lamont, "Visualizing Particle Swarm Optimization - Gaussian Particle Swarm Optimization," *Proceedings of the IEEE Swarm Intelligence Symposium,* pp. 198-204, 2003.

[45] X. Yao and Y. Liu, "Fast Evolutionary Programming," *Proceedings of the Fifth Annual Conerence on Evolutionary Programming,* pp. 451-460, 1996.

[46] X. Yao, Y. Liu and G. Liu, "Evolutionary Programming Made Faster," *IEEE Transactions on Evolutionary Computation,* vol. 2, no. 3, pp. 82-102, 1999.

[47] M. Løvberg, T. Rasmussen and T. Krink, "Hybrid Particle Swarm Optimiser with Breeding and Subpopulation," *Proceedings of the Genetic and Evolutioary Computation Conference,* pp. 469-476, 2001.

[48] B. Al-Kazemi and C. Mohan, "Multi-phase Generalization if the Particle Swarm Optimization Algorithm," *Proceedings of the IEEE Congress on Evolutionary Computation,* pp. 489-494, 2002.

[49] T. Krink and M. Løvberg, "The Life-Cycle Model: Combining Particle Swarm Optimization, Genetic Algorithm and Hill Climbers," *Proceedings of the Parallel Problem Solving from Nature Conference, Lecture Notes in Computer Science,* vol. 2439, pp. 621-630, 2002.

[50] K. Schwaber and J. Sutherland, The Scrum Guide: The definitive Guide to Scrum, The rules of the Game, 2011.

[51] A. Gephi, "Gephi: Makes graphs handi," Association Gephi, [Online]. Available: https://gephi.org/.

[52] T. Reenskaug, "THING-MODEL-VIEW-EDITOR: an Example from a planningsystem," Palo Alto, 1979.

[53] M. C. Robert, "The Open-Closed Principle," *C++ Report,* p. 1, 1996.

[54] M. L. Braun, J. Schaback, M. L. Jugel and N. Oury, "Linear Algebra for Java," [Online]. Available: http://mikiobraun.github.io/jblas/.

[55] y. Richet, "JMATHTOOLS," [Online]. Available: http://jmathtools.berlios.de/doku.php?id=start.

[56] A. C. Oliver, G. Stampoultzis, A. Sengupta, R. Klut and D. Fisher, "Apache POI: The Java library for Microsoft document," The Apache Software Foundation, [Online]. Available: http://poi.apache.org/.

[57] C. R. Hughes, "Modeling Movie Release Strategies," San Francisco, 2012.

[58] A. Inc., "Multimethod simulation software and solution," Anylogic, [Online]. Available: http://www.anylogic.com/.

[59] F. Collins and D. Gallas, "A nw five-year plan for the U.S. Human Genome Project," *Science,* vol. 262, pp. 43-46, 1993.

[60] J. Claverie, "Computational methods for the identification of genes in vertebrate genomic sequences," *In Human Molecular Genetics,* pp. 1735-1744, 1997.

[61] H. Douzono, S. Hara and Y. Noguchi, "A Design Method of DNA chips for SNP Analysis Using Self Organising Maps," in *Proceedings of IEEE International Joint Conference on Neural Network*, Washington DC, 2001.

[62] D. F. Feng and R. F. Doolittle, "Progressive sequence alignment of amino acid sequences and construction of phylogenetic trees from them," *Methods in Enzymology,* p. 266, 1996.

[63] T. Kristensen, "A Neural Network Approach to Hyphenating Norwegian," in *Proceedings of IEEE International Joint Conference on Neural Networks*, Como, 2000.

[64] T. Kristensen and D. Langmyhr, "Two Regimes for Computer Hyphenation – a Comparison.," in *Proceedings of IEEE International Joint Conference on Neural Networks*, Washington DC, 2001.

[65] T. Kristensen and R. Patel, "Edge Detecting in a Lateral Inhibition Network," in *Proceedings of IEEE World Congress on Computational Intelligence*, Honolulu, 2002.

[66] C. F. Allex, J. W. Shavlik and F. R. Blattner, "Neural Network Input Representations that Produce Accurate Consensus Sequences fram DNA Fragment Assemblies," *Bioinformatics,* vol. 15, 199.

[67] R. Nusinov, "Strong Preferences in Nucleotide Sequences of DNA Geometry," *Journal of Molecular Evolution,* vol. 20, 1984.

[68] T. Kristensen and F. Guillaume, "Classification of DNA sequences by a MLP and a SVM Network," in *Biocomp'13 – International Conference Bioinformatics and Computational Biology*, Las vegas, 2013.

[69] T. M. Mitchell, Machine Learning, Carnegie Mellon University: McGRAW-HILL internation editions, 1997.

[70] "Java Neural Network Framework Neuroph," [Online]. Available: http://neuroph.sourceforge.net/index.html.

[71] "LIBSVM -- A Library for Support Vector Machines," [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[72] X. Guo, J. Yang, C. Wu, C. Wang and Y. Liang, "A novel LS-SVMs hyper parameter selection baesd on particle swarm optimization," *Neurocomputing,* no. 71, pp. 3211-3215, 2008.

[73] S.-W. Lin, K.-C. Ying, S.-C. Chen and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines," *Expert Systems with Applications,* no. 35, pp. 1817-1824, 2008.

[74] C.-L. Huang and J.-F. Dun, "A distributed PSO-SVM hybrid system with feature selection and parameter optimization," *Applied Soft Computing,* no. 8, pp. 1381-1391, 2008.

[75] J. C. Platt, "Fast Training of Support Vector Machines using Sequential Minimal Optimization," Microsoft Research, Redmond, 2000.

[76] L. Fu and E. Medico, "FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data.," *BMC bioinformatics,* vol. 1, no. 8, p. 3, 2007.

[77] J. Platt, "Sequential Minimal Optimization: A fast Algorithm for Training Support Vector Machine," 1998.

[78] A. Ben-Hur, D. Horn, H. T. Siegelmann and V. Vapnik, "Support Vector Clustering," *Journal of Machine Learning Research 2,* pp. 125-137, 2001.

[79] P. Software, "Spring," Pivotal Software, 2013. [Online]. Available: http://spring.io/.

[80] M. Wooldridge, An introduction to Multi Agent Systems, Liverpool: Wiley, 2009.

[81] S. Russel and P. Norvig, Artificial Intelligence A modern approach, Pearson, 2010.

[82] "JADE - Java Agent Development Framework," Telecom Italia, Motorola, Whitestein Technologies AG, Profactor GmbH, France Telecom R&D, 6 12 2013. [Online]. Available: http://jade.tilab.com/.

[83] M. Pernollet, "jzy3d," [Online]. Available: http://jzy3d.org/index.php.

[84] O. s. community, "JFreeChart," Object Refinery Limited, [Online]. Available: http://www.jfree.org/jfreechart/.

Appendix

Redundancy of the genetic code:

The following table lists the different codons, their corresponding molecule and their index in our frequency vector. The color is just to visualize the redundancy.

final index	Index	Codons	Translation
0	0	Phe/F (Phenylalanine)	TTT
0	1	Phe/F (Phenylalanine)	TTC
1	2	Leu/L (Leucine)	TTA
1	3	Leu/L (Leucine)	TTG
1	4	Leu/L (Leucine)	CTT
1	5	Leu/L (Leucine)	CTC
1	6	Leu/L (Leucine)	CTA
1	7	Leu/L (Leucine)	CTG
2	8	Ile/I (Isoleucine)	ATT
2	9	Ile/I (Isoleucine)	ATC
2	10	Ile/I (Isoleucine)	ATA
3	11	Met/M (Methionine) START	ATG
4	12	Val/V (Valine)	GTT
4	13	Val/V (Valine)	GTC
4	14	Val/V (Valine)	GTA
4	15	Val/V (Valine)	GTG
5	16	Ser/S (Serine)	TCT
5	17	Ser/S (Serine)	TCC
5	18	Ser/S (Serine)	TCA
5	19	Ser/S (Serine)	TCG
6	20	Pro/P (Proline)	CCT
6	21	Pro/P (Proline)	CCC
6	22	Pro/P (Proline)	CCA
6	23	Pro/P (Proline)	CCG
7	24	Thr/T (Threonine)	ACT
7	25	Thr/T (Threonine)	ACC
7	26	Thr/T (Threonine)	ACA
7	27	Thr/T (Threonine)	ACG
8	28	Ala/A (Alamine)	GCT
8	29	Ala/A (Alamine)	GCC
8	30	Ala/A (Alamine)	GCA

8	31	Ala/A (Alamine)	GCG
9	32	Tyt/Y (Tyrosine)	TAT
9	33	Tyt/Y (Tyrosine)	TAC
10	34	STOP (Ochre)	TAA
10	35	STOP(Amber)	TAG
11	36	His/H (Histidine)	CAT
11	37	His/H (Histidine)	CAC
12	38	Gln/Q (Glutamine)	CAA
12	39	Gln/Q (Glutamine)	CAG
13	40	Asn/N (Asparagine)	AAT
13	41	Asn/N (Asparagine)	AAC
14	42	Lys/K (Lysine)	AAA
14	43	Lys/K (Lysine)	AAG
15	44	Asp/D (Aspartic acid)	GAT
15	45	Asp/D (Aspartic acid)	GAC
16	46	Glu/E (Flutamic acid)	GAA
16	47	Glu/E (Flutamic acid)	GAG
17	48	Cys/C (Cysteine)	TGT
17	49	Cys/C (Cysteine)	TGC
10	50	STOP (Opal)	TGA
18	51	Trp/W (Tryptophan)	TGG
19	52	Arg/R (Arginine)	CGT
19	53	Arg/R (Arginine)	CGC
19	54	Arg/R (Arginine)	CGA
19	55	Arg/R (Arginine)	CGG
20	56	Ser/S (Serine)	AGT
20	57	Ser/S (Serine)	AGC
19	58	Arg/R (Arginine)	AGA
19	59	Arg/R (Arginine)	AGG
21	60	Gly/G (Glycine)	GGT
21	61	Gly/G (Glycine)	GGC
21	62	Gly/G (Glycine)	GGA
21	63	Gly/G (Glycine)	GGG

Table 0.1: Genetic code