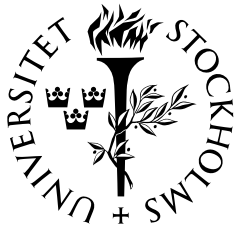

SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Unconstrained Particle Swarm Optimizer for Variable Weighting in Soft Projected Clustering of High-Dimensional Data

av

Kristoffer Vinell

2010 - No 13

Unconstrained Particle Swarm Optimizer for Variable Weighting in Soft Projected Clustering of High-Dimensional Data

Kristoffer Vinell

Självständigt arbete i tillämpad matematik 30 högskolepoäng, avancerad nivå

Handledare: Yishao Zhou

2010

Abstract

Due to the increasing volumes of stored data arising in various fields of business and research, the demand for efficient data analysis tools have skyrocketed in recent years. A popular approach well-suited for tackling high-dimensional data in particular is soft projected clustering, which aims at partitioning the data objects into disjoint subsets. Soft projected clustering is particularly interesting from a mathematical viewpoint, since the clustering process is cast in the form of a nonlinear optimization problem. However, most existing algorithms involve a large number of bound and equality constraints, which severely restrict the performance of the optimization method employed.

In this thesis, a new soft projected clustering algorithm called UPSOVW is developed to overcome these issues. It uses an objective function that enables an unconstrained search procedure by eliminating redundant bound constraints, and employs a particle swarm optimizer in quest for a global optimum. We formally prove that the bound constraints can be omitted without loss of generality, and conduct a stability analysis that provides guidelines for suitable parameter settings in the algorithm. Finally, we compare UPSOVW to an existing algorithm on a number of synthetic high-dimensional data sets.

Acknowledgements

I would like to thank my supervisor, Yishao Zhou, of the Department of Mathematics at Stockholm University, for taking an interest in my work at an early stage and for encouraging further research. For this I am very grateful. I would also like to thank Yanping Lu of the Department of Computer Science at University of Sherbrooke, for providing the source code for the PSOVW algorithm and guidelines for how to generate synthetic data sets.

Contents

1 Introduction

There is a growing trend around the globe to collect and store more and more data. Companies store customer data to make better marketing decisions, the Internet grows like clockwork and governmental institutions like NASA collect terabytes of cosmic data on a daily basis. The increasing volumes of data require efficient methods of handling and analyzing these enormous amounts. Ideally, data becomes information, and information becomes knowledge. This is however far from reality in many cases.

As the demand for these methods has increased in recent years, several new fields have emerged. *Data mining*, which is one of them, is the process of extracting patterns from data. *Anomaly detection*, on the other hand, deals with detecting patterns in a given data set that do not conform to an already established normal behavior. These have applications in various fields such as credit card fraud detection ([1]), network security ([2]) and insurance company customer prediction ([3]), to name a few. Although these fields may seem unrelated at first sight, there is a striking resemblance between them from a theoretical viewpoint, in that they all rely on the accuracy and efficiency of data mining and anomaly detection algorithms. For instance, in an intrusion detection application managing network surveillance, these aspects are crucial. Anomalies must be rejected quickly yet accurately, so that harmless network usage is not mistaken for anomalous behaviour.

1.1 Clustering

A common task arising in many of the above applications is *clustering*. Clustering deals with partitioning a collection of data objects into a number of disjoint subsets (clusters). The aim is to divide the given data set such that objects in the same cluster are similar to each other with respect to some predefined similarity measure, whereas objects in different clusters are dissimilar. The choice of a suitable similarity measure is an important and difficult problem in its own right, since it is application-specific and greatly influences the cluster quality.

An example taken from insurance company customer prediction may motivate the use of a clustering method and may also illuminate some of the design issues therein. Suppose that an insurance company launches a new insurance policy. The company is then interested in separating their customers into groups to predict who would be interested in buying such an insurance ([4]). In order to make the predictions more reliable, each customer has a data record containing several variables valuable to the insurance company (age, sex, marital status, income etc.). The number of variables suitable to consider in such an application is often in the hundreds or even thousands, depending on the insurance of interest.

The intuitive approach in clustering high-dimensional data of this kind

would be to use a similarity measure based on a metric, such as the Euclidean distance measure. Any two data objects that are close to each other in the given metric space would fall into the same cluster. However, as the number of dimensions increases, data becomes very sparse and distance measures in the whole dimension space become pointless. This phenomenon is often referred to as *the curse of dimensionality* and will be encountered many times in the following, since high-dimensional data clustering is at the core of this work.

An immediate consequence of increasing dimensionality is that clusters of high-dimensional data are usually embedded in lower-dimensional subspaces, which makes clustering a difficult task. As a matter of fact, some dimensions may be irrelevant or redundant for clusters and different sets of dimensions may be relevant for different clusters. Consequently, clusters should usually be searched for in subspaces of dimensions rather than the whole dimension space.

Three categories of clustering methods have been considered following this approach (as defined in [5]). The first, *subspace clustering*, aims at finding all subspaces where clusters can be identified (see for instance [6], [7], [8]). Thus, algorithms that fall into this category are dedicated to finding all clusters in all subspaces. The second, *projected clustering*, aims at dividing the data set into disjoint clusters (see for instance [9], [10]). The third type, *hybrid clustering*, falls in between the two. These algorithms are in general intended to find clusters that may overlap (see for instance [11]). On the other hand, these algorithms do not aim at finding all clusters in all subspaces. In fact, some of the hybrid algorithms only compute interesting subspaces rather than final subspace clusters. The retrieved subspaces can then be processed by applying full-dimensional algorithms to these projections.

In recent years, a specialized version of projected clustering has been developed, called *soft projected clustering*. This group of methods identify clusters by assigning an optimal variable weight vector to each cluster (see for instance [12], [13], [14], [15]). The clustering is carried out by minimizing an objective function iteratively by finding better and better variable weight configurations. The objective function is similar to the k-means objective function introduced in [16]. Although the cluster membership of a data object is determined by considering the whole variable space, the similarity between each pair of objects is based on weighted variable differences. Different attributes are just differently weighted, but all attributes contribute to the clustering. However, in order to define a suitable objective function, the number of clusters k must be known beforehand, which is one of the drawbacks of this type of clustering. On the other hand, once the objective function is set, the problem at hand is a well-defined optimization problem, which can be tackled by any suitable optimization technique. This is a valuable property from a mathematical viewpoint, since it divides the clus-

tering issues into the problem of firstly choosing a suitable objective function followed by employing an efficient optimization strategy.

1.2 Disposition

The rest of the thesis is organized as follows. Section 2 gives some further insight into the mechanics of soft projected clustering and presents a recent method in this field, namely the PSOVW algorithm. Section 3 reviews some previous related work on the variable weighting problem in soft projected clustering and briefly presents three well-known clustering algorithms other than PSOVW. The particle swarm optimization technique is covered in section 4, which is the foundation of the search strategy employed in PSOVW. Section 5 gives a detailed description of the PSOVW algorithm. In section 6, the UPSOVW algorithm is proposed, which is the main contribution of the thesis. It is similar but not identical to PSOVW, in that it employs a different search strategy. This section also presents the main theorem which confirms the efficiency of the new search strategy. The main theorem and the analysis of UPSOVW that follows in section 7 covers the theoretical aspect of the thesis. In this section, an extensive literature study reveals some important previous work concerning the stability of PSO methods, which is followed by some new ideas for such an analysis with an emphasis on UPSOVW. In section 8, the PSOVW and UPSOVW algorithms are run on a variety of high-dimensional data sets and the experimental results are presented. The final section draws a number of conclusions and suggests directions for future work.

2 Problem statement

Soft projected clustering is a tractable yet powerful clustering method for several reasons. A few of these have already been pointed out in the previous section. The advantages become even more apparent in a recent article by Lu, Wang, Li and Zhou, where a novel approach to the variable weighting problem in soft projected clustering of high-dimensional data is proposed ([12]).

The objective of the soft projected clustering algorithm is to partition a set of n data objects with m dimensions into k clusters. Note that, as with any soft projected clustering algorithm, the number of clusters must be known beforehand. Lu et al. proposed the problem of minimizing an objective function $F\colon \mathbf{R}^{k\times m} \to \mathbf{R}$ defined as

$$F(W) = \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{m} u_{l,i} \cdot \left(\frac{w_{l,j}}{\sum_{s=1}^{m} w_{l,s}}\right)^{\beta} \cdot d(x_{i,j}, z_{l,j})$$

subject to the constraints

$$\begin{cases} 0 \leq w_{l,j} \leq 1, & 1 \leq l \leq k, 1 \leq j \leq m \\ \sum_{l=1}^{k} u_{l,i} = 1, & u_{l,i} \in \{0,1\}, 1 \leq i \leq n. \end{cases}$$

The variable weight matrix W contains one entry for each dimension on each cluster, where each entry appears several times in the objective function. The variable $x_{i,j}$ denotes the value of data object i on dimension j, and $z_{l,j}$ is the centroid of cluster l on dimension j. A cluster centroid doesn't necessarily have to be one of the data objects. In fact, they rarely are, and should rather be regarded as artificial objects steering the algorithm in the right direction. The function d is a distance function measuring the similarity between a data object and a cluster centroid, e.g. the Euclidean distance. The indicator variable $u_{l,i}$ is the membership of data object i in cluster l. It appears together with its equality constraint to ensure that each object belongs to a unique cluster. Included in the objective function is also a positive constant β that magnifies the importance of variables with large weights, and lets them influence the separation of relevant dimensions from irrelevant ones. A large value of β makes the objective function more sensitive to changes in the weights. Generally, variables that share a strong correlation with a cluster obtain large weights, which implies that these variables play a strong role in the identification of data objects in the cluster. Conversely, irrelevant variables in a cluster obtain small weights. The computation of the membership of a data object in a cluster consequently depends on the variable weights as well as the cluster centroids.

As mentioned, the performance of soft projected clustering is greatly influenced by the objective function and the search strategy employed. The

objective function determines the cluster quality, whereas the search strategy has an impact on whether the optimum of the objective function can be found. The cluster quality is closely related to the clustering accuracy, which is referred to as the percentage of the data objects that are correctly identified by the algorithm. This is in turn closely related to the clustering variance, which is defined as the variance of the clustering accuracy in between runs on the same data set. It can be thought of as an indicator to how robust the method is. Apart from these issues, the computational complexity of the optimization method should also be taken in consideration when deciding on a search strategy.

Lu et al. use a method that stems from a large class of optimization techniques called *particle swarm optimization* (PSO) ([17]). This method along with the objective function are the core elements of their clustering algorithm named PSOVW (Particle Swam Optimizer for Variable Weighting). The objective of this thesis is to present an algorithm, similar to PSOVW, by altering either the objective function or the search strategy. The algorithm should yield faster convergence while maintaining high cluster quality.

3 Related work

This section covers some related work in the field of soft projected clustering. Three algorithms are presented, namely LAC ([13]), the W-k-means algorithm ([14]) and EWKM ([15]). These algorithms are all quite similar to PSOVW, so the same notation will be used throughout this work for convenience.

3.1 LAC

The LAC algorithm (Locally Adaptive Clustering) was recently proposed by Domeniconi et al., which computes the clusters by minimizing the following objective function:

$$\begin{aligned} F(w) &= \sum_{l=1}^{k}\sum_{j=1}^{m}\left(w_{l,j}\cdot X_{l,j} + h\cdot w_{l,j}\cdot \log w_{l,j}\right), \\ X_{l,j} &= \frac{\sum_{i=1}^{n} u_{l,i}\cdot (x_{i,j}-z_{l,j})^2}{\sum_{i=1}^{n} u_{l,i}} \end{aligned}$$

subject to

$$\begin{cases} \sum_{j=1}^{m} w_{l,j} = 1, & 0 \le w_{l,j} \le 1, \quad 1 \le l \le k \\ \sum_{l=1}^{k} u_{l,i} = 1, & u_{l,i} \in \{0,1\}, \quad 1 \le i \le n. \end{cases}$$

Here, $X_{l,j}$ is the squared variance of cluster l along dimension j. Notably, the constraints in LAC bear a resemblance to the ones used in PSOVW. However, not only do they enforce upper and lower bounds on the variable weights, but there is also an equality constraint on the variable weights.

In the following description of the algorithm U, Z and W represent the cluster membership matrix of data objects, the cluster centroids matrix and the dimensional weights matrix, respectively. The following formulas are used to update the cluster membership and cluster centroids:

$$\begin{cases} u_{l,i} = 1, & \text{if } \sum_{j=1}^{m} w_{l,j}\cdot (x_{i,j}-z_{l,j})^2 \le \sum_{j=1}^{m} w_{q,j}\cdot (x_{i,j}-z_{q,j})^2, \\ & \text{for } 1 \le q \le k \\ u_{q,i} = 0, & \text{for } q \ne l, \end{cases} \tag{3.1}$$

$$z_{l,j} = \frac{\sum_{i=1}^{n} u_{l,i}\cdot x_{i,j}}{\sum_{i=1}^{n} u_{l,i}}, \quad \text{for } 1 \le l \le k, 1 \le j \le m, \tag{3.2}$$

$$w_{l,j} = \frac{\exp\left(-X_{l,j}/h\right)}{\sum_{s=1}^{m}\exp\left(-X_{l,s}/h\right)}, \quad \text{for } 1 \le l \le k, 1 \le j \le m. \tag{3.3}$$

Given these formulas, the LAC algorithm operates as follows:

Initialization:

Select k well-scattered data objects as k initial centroids.

Set initial weights $w_{l,j} = 1/m$, for each dimension in each cluster.

Repeat:

Update the cluster membership matrix U by (3.1).

Update the cluster centroids matrix Z by (3.2).

Update the dimension weights W by (3.3).

Until: (no change in the centroids' coordinates is observed, or the number of function evaluations reaches a specified threshold).

The parameter h is chosen to maximize (or minimize) the influence of $X_{l,j}$ on $w_{l,j}$. In practice, the tuning of h is problem-specific and should therefore be empirically determined, which is a difficult problem in its own right. It should be noted that the LAC technique is centroid-based, like PSOVW, because weightings depend on the centroid. The computed weights are used to update the cluster membership matrix, and therefore the centroids' coordinates.

3.2 W-k-means

In 2005, Huang, Ng, Rong and Li proposed a different objective function and presented an algorithm known as the W-k-means algorithm ([14]). The objective function is given by

$$F(w) = \sum_{l=1}^{k} \sum_{i=1}^{n} \sum_{j=1}^{m} u_{l,i} \cdot w_j^{\beta} \cdot d\left(x_{i,j}, z_{l,j}\right)$$

subject to

$$\begin{cases} \sum_{j=1}^{m} w_j = 1, & 0 \leq w_j \leq 1, \quad 1 \leq l \leq k \\ \sum_{l=1}^{k} u_{l,i} = 1, & u_{l,i} \in \{0,1\}, \quad 1 \leq i \leq n. \end{cases}$$

The most notable difference compared to LAC and PSOVW is that there is only one weight variable for each dimension, whereas LAC and PSOVW use one variable for each dimension for each cluster. The immediate consequence of this design is that the objective function in W-k-means measures the sum of the within-cluster distances along variable subspaces rather than over the entire variable space. The variable weights are enforced by the same

equality and bound constraints used in LAC. W-k-means also operates much like LAC except that the weights are updated by

$$w_j = \begin{cases} 0, & \text{if } D_j = 0 \\ \left(\sum_{s=1}^{m} \left[\frac{D_j}{D_s} \right]^{1/(\beta-1)} \right)^{-1}, & \text{if } D_j \neq 0, \end{cases}$$

where

$$D_j = \sum_{l=1}^{k} \sum_{i=1}^{n} u_{l,i} \cdot d\left(x_{i,j}, z_{l,j}\right),$$

and the cluster memberships are updated by

$$\begin{cases} u_{l,i} = 1, & \text{if } \sum_{j=1}^{m} w_j^{\beta} \cdot d\left(x_{i,j}, z_{l,j}\right) \leq \sum_{j=1}^{m} w_j^{\beta} \cdot d\left(x_{i,j}, z_{q,j}\right), \\ & \text{for } 1 \leq q \leq k \\ u_{q,i} = 0, & \text{for } q \neq l. \end{cases}$$

The W-k-means algorithm is largely influenced by the k-means algorithm ([16]). A weight is assigned to each dimension and the algorithm aims at minimizing the sum of all the within-cluster distances in the same subset of dimensions. A large weight is reflected by a small sum of within-cluster distances in a dimension, implying that the dimension contributes more to the cluster than a dimension with a small weight tied to it.

Moreover, both the objective function and the updating procedure is largely influenced by the value of the parameter β. In the analysis by Huang et al., it is proposed that one should choose either $\beta < 0$ or $\beta > 1$ for best performance. This decision is especially crucial when dealing with high-dimensional data. Their analysis also shows that the algorithm converges to a local minimal solution in a finite number of iterations. Despite this, Lu et. al ([12]) argue that the W-k-means algorithm does not employ an efficient search strategy, which is the major drawback of the algorithm. As a consequence, clusters embedded in different subsets of variables are often left unexplored by the algorithm. Due to the fact that it assigns a unique weight to each dimension for all clusters, the W-k-means algorithm is in general not suited for high-dimensional data clustering where each cluster has its own subset of relevant dimensions.

3.3 EWKM

In an attempt to improve the W-k-means algorithm, Jing, Ng and Huang introduced the entropy weighting k-means algorithm, or EWKM, for short ([15]). It operates in a similar fashion to W-k-means, but is better suited for dealing with high-dimensional clustering since it assigns one weight to each variable for each cluster. The objective function is also adjusted to cope with

the numerous challenges that may be encountered in high-dimensional clustering, as it accounts for both the within-cluster dispersion and the weight entropy. The concept of entropy has actually been encountered before, as it is included in the LAC objective function. The objective function utilized in EWKM is given by

$$F(w) = \sum_{l=1}^{k} \left[\sum_{i=1}^{n} \sum_{j=1}^{m} u_{l,i} \cdot w_{l,j} \cdot (x_{i,j} - z_{l,j})^2 + \gamma \cdot \sum_{j=1}^{m} w_{l,j} \cdot \log w_{l,j} \right]$$

subject to

$$\begin{cases} \sum_{j=1}^{m} w_{l,j} = 1, & 0 \leq w_{l,j} \leq 1, \quad 1 \leq l \leq k \\ \sum_{l=1}^{k} u_{l,i} = 1, & u_{l,i} \in \{0,1\}, \quad 1 \leq i \leq n. \end{cases}$$

The constraints of the above function are exactly the ones that need to be met in the LAC algorithm. The weights are updated according to

$$w_{l,j} = \frac{\exp\left(-D_{l,j}/\gamma\right)}{\sum_{s=1}^{m} \exp\left(-D_{l,s}/\gamma\right)}, \quad D_{l,j} = \sum_{i=1}^{n} u_{l,i} \cdot (x_{i,j} - z_{l,j})^2 .$$

The idea behind the algorithm is closely related to the weight entropy concept. In subspace clustering, a decrease of weight entropy in a cluster is reflected by an increase of certainty of a subset of dimensions. Hence, the objective function should simultaneously minimize both the within-cluster dispersion as well as the weight entropy in order to stimulate more dimensions to contribute to the formation of a cluster. This minimization process is heavily influenced by the parameter γ. Therefore, γ should be tuned such that the entropy component has the desired effect on the objective function and the search strategy.

3.4 Common features

Notably, the LAC, the W-k-means and EWKM algorithms are similar in several ways. Firstly, they all converge to a local optimum in a finite number of iterations. Moreover, it can be shown that the computational complexity of the three algorithms is $O(mnkT)$, where T is the number of iterations and m, n, k are the number of dimensions, the number of data objects and the number of clusters, respectively. The computational complexity thus increases linearly as the number of dimensions, the number of data objects or the number of clusters increase, which is considered as a relatively small price compared to many other algorithms appearing in soft projected clustering.

However, since they are all derived from the k-means algorithm, they have a few problems in common. For instance, their respective objective

functions are not satisfactory in some ways, many of which have been mentioned already. Another drawback is that the cluster quality generated by the algorithms is highly sensitive to the choice of the initial cluster centroids, which was recently shown by experiments carried out by Lu et al. ([12]). Further, all the three algorithms utilize local search strategies to optimize their objective functions, which severely restrict the search space and increase the risk of getting trapped in local optima. As a result, good convergence speed is achieved at the expense of cluster quality. Beside poor design of the search strategies, all of the above algorithms require a user-defined parameter that is empirically set and not easily tuned.

4 Particle swarm optimization

Particle swarm optimization is an optimization technique especially suited for optimizing continuous nonlinear functions. It was introduced by Kennedy and Eberhart in 1995 ([17]) and has been an increasingly popular optimization method in several fields since, soft projected clustering being no exception.

Swarm behaviour is a common phenomenon in nature, and can be seen in bird flocks, fish schools as well as in ant colonies and among mosquitoes. As can be seen, these animal groups all follow an ordered structure among them, each organism contributing to a uniform choreography. Although the formation may change shape and direction, they seem to move as a coherent unit. This observation is especially apparent during the search for food and is often referred to as *swarm intelligence*.

In PSO, social behaviour of this kind is simulated in quest for an optimal solution. Each solution to the given optimization problem is regarded as a particle in the feasible space, and each particle contribute to the swarm. Each particle has a position, which is usually a feasible solution to the problem given, and a velocity. The positions are evaluated by a fitness function supplied by the user, i.e. the objective function. During each iteration, the velocities, and consequently the positions of the particles, are updated by a formula heavily influenced by the swarms best positions found so far. Because of the dependence on historical best positions, the particles successively fly towards better regions of the search space, influenced by their own experiences as well as the experiences of the whole swarm. By this fact, the population converges to an optimal or near-optimal solution.

The velocity and position of the ith particle are updated as follows:

$$v_i(t+1) = \lambda v_i(t) + \phi_p r_p \left[p_i(t) - x_i(t)\right] + \phi_g r_g \left[g(t) - x_i(t)\right], \quad (4.1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1), \quad (4.2)$$

where x_i is the position of the ith particle, v_i is its corresponding velocity and p_i is its personal best position found so far according to the fitness function (which is often denoted $pBest$ in the literature). The variable g is the global best position retrieved up to now by the whole swarm (often denoted $gBest$), i.e. the best value among the personal best positions. The parameter λ is called *inertia weight*, which is tuned to control the influence of the global versus the local search capabilities. The parameters ϕ_p and ϕ_g are *acceleration factors*, which are often referred to as the *cognitive parameter* and the *social parameter*, respectively. They have an impact on how much the particles should be biased toward the personal best and the global best. The method is given a stochastic nature by r_p and r_g, which are random numbers uniformly distributed in the range $[0, 1]$. These are updated at each iteration.

Before running the algorithm, the number particles of the population must be set, apart from tuning the parameters introduced above. The swarm size is largely dependent on the objective function, i.e. the anticipated difficulty of the problem at hand, and the search space, i.e. the dimensionality and constraints, if any. However, a value in between 10 and 50 particles should do it for most applications. One should keep in mind that there is a trade-off related to this decision. A large population size increases the algorithm's chances of finding a global optimum, but the swarm will require more iterations to converge.

The PSO algorithm for a minimization problem is summarized as follows, given the objective function f:

Initialization:

Randomly initialize the position and velocity swarms, X and V.

Initialize the personal best positions by setting $p_i = x_i$, for each particle in X.

Store the swarm of personal bests as P.

Evaluate the fitness of each particle in X by the objective function f.

Initialize the global best position by setting g to the best particle in P:

$$g = \arg\min_i f(p_i)$$

Repeat:

Update for each particle in V according to (4.1).

Update for each particle in X according to (4.2).

Update for each particle in P according to

$$p_i = \begin{cases} x_i, & \text{if } f(x_i) < f(p_i) \\ p_i, & \text{otherwise} \end{cases}$$

Update the global best according to

$$g = \arg\min_i f(p_i)$$

Until: (the objective function reaches a global minimum value, or the number of function evaluations reaches a specified threshold).

In PSO, convergence can always be guaranteed because it has memory. While the swarm may change during each iteration, the best position

found so far is always kept, which guarantees convergence of the algorithm. However, although PSO always converges, it may not be guaranteed that the global optimum is reached. In some circumstances, there is *stagnation*, which is said to occur if the personal bests, and consequently the global best, do not change over some iterations ([18]). PSO can even suffer from *premature convergence* in severe cases, where the swarm gets trapped in local optima. In other cases, the swarm will unavoidably slow down as it approaches optima. This phenomenon occurs when the swarm is to one side of the optimum in scope and is moving as a coordinated entity down the function gradient.

Needless to say, the outcome of the algorithm is largely dependent on the diversity of the swarm, i.e. how well the particles are scattered in the search space. During initialization, the particles are randomly spread in the search space. It is important that the particles are well spread, because at the first iteration, the particles are influenced only by the global best position in the swarm and not by their personal bests (since any personal best corresponds to the particle itself after initialization). This is clear from formula (4.1), where the factor $(p_i - x_i)$ is zero at the first iteration.

Generally, one is looking for a balanced *exploration* and *exploitation* capability, where swarm diversity plays a strong role. When the particles are away from good enough solutions and are diverse enough, the algorithm should have more exploitation capability than exploration capability, meaning that the swarm should focus more on the converging process. On the other hand, when the particles are clustered and are away from good enough solutions, the swarm should have more exploration than exploitation capability, that is, it should be more in the diverging process. It is possible to maintain a level of diversity throughout the run by the use of certain strategies. For instance, Parsopoulos and Vrahatis utilized repulsion to keep particles away from previously located optima ([19]). Blackwell and Bentley, on the other hand, treated the particles as charged, as they incorporate electrostatic repulsion between the particles ([20]). These particles mutually repel each other but eventually start to converge, following an orbit surrounding a core of "neutral" particles. A similar behaviour can be seen among atoms. Charged swarms of this kind can detect and respond to changes in the optimum within their orbit and are therefore able to observe relatively drastic changes. None of these methods are however used in the original implementation of PSO. Luckily, the swarm diversity can be controlled to some extent by the use of proper parameter settings; the values of the inertia weight and the acceleration factors, as well as by the use of *velocity clamping*.

4.1 Parameter settings and velocity clamping

The inertia weight, λ, was actually not included in the original version of PSO. Instead, one used the formula

$$v_i(t+1) = v_i(t) + \phi_p r_p \left[p_i(t) - x_i(t)\right] + \phi_g r_g \left[g(t) - x_i(t)\right],$$

for the velocity dynamics (which is, in effect, the formula obtained by setting $\lambda = 1$ in (4.1)). Because the inertia weight was omitted, the acceleration factors were inclined to play a much stronger role in keeping the swarm diverse and at the same time maintaining convergence to an optimum. As can be seen, the values $\phi_p r_p$ and $\phi_g r_g$ introduce a kind of stochastic weighting in the system, where the acceleration factors determine the magnitude of the stochastic influence on the algorithm. The cognitive parameter, ϕ_p, determines the random force in the direction of the personal best position, whereas the social parameter, ϕ_g, determines the random force in the direction of the global best position.

The values of the acceleration factors have a great impact on the behaviour of PSO. Low values tend to let particles move far away from target regions before being restrained, while high values result in abrupt movement toward, or past, target regions. An interesting point is that we can interpret the components $\phi_p r_p(p_i - x_i)$ and $\phi_g r_g(g - x_i)$ as attractive forces in a spring-mass system with springs of random stiffness. In this setting, the motion of a particle can be approximated by applying Newton's second law. Then, the entities $\phi_p/2$ and $\phi_g/2$ represent the mean stiffness of the springs attracting a particle. With this interpretation, it is clear that the choice of the acceleration factors can make the PSO more or less "tense" and possibly even unstable, with particles speeding without control. This can be more or less harmful to the search procedure and happened frequently in the early days of PSO, when the parameter values $\phi_p = \phi_g = 2$ were used without further insight in the stability issues.

Eventually, researchers dealt with this problem, not by changing the values of the parameters, but by introducing a concept known as *velocity clamping*. It first appeared in the work by Eberhart, Simpson and Dobbins ([21]) just about a year after the PSO algorithm had been proposed. The technique introduces a maximum velocity threshold, $v_{\max}$, that may not be traversed by any velocity on any dimension. If an update would result in a dimension exceeding the threshold, then the velocity on that dimension is limited to $v_{\max}$. This ensures that each dimension for each particle velocity is kept within the range $[-v_{\max}, v_{\max}]$. The threshold $v_{\max}$ is therefore an important parameter, since it determines the resolution, or fineness, with which regions between the present position and the targets (personal best and global best positions) are searched. If $v_{\max}$ is too high, particles may fly past better regions. On the other hand, if $v_{\max}$ is too low, particles might not be able to fully explore regions beyond the present local scope. It is even

plausible that the particle could become trapped in local optima, without any chance of moving far enough to reach a better position in the search space.

Because the parameter $v_{\max}$ appears to influence the interplay between exploration and exploitation, it should be chosen with care. Although the optimal value for the parameter is problem-specific, there are no general guidelines and the user must in most cases discover a suitable value empirically. However, once the parameter is set, the initialization of the particle velocities are easily determined, as they are simply randomly scattered in the range $[-v_{\max}, v_{\max}]$. If velocity clamping is not applied, then the user must come up with a different scheme to initialize them. This decision should be based on the size of the search space, since there is no point in setting $v_{\max}$ to a value that allows the particles to fly outside the search space.

In an attempt to eliminate the need for velocity clamping, the concept of inertia weight was coined and introduced a few years later by Shi and Eberhart ([22]). It has improved the performance of PSO in numerous applications and may be interpreted from a scientific viewpoint, much like the acceleration factors. By considering $\phi_p r_p(p_i - x_i) + \phi_g r_g(g - x_i)$ as an external force, F_i, applied to a particle, then the change in a particle's velocity (i.e. the particle's acceleration) can be expressed as $\Delta v_i = F_i - (1 - \lambda)v_i$. Thus, the factor $1 - \lambda$ is in effect a friction coefficient. In fluid dynamics, the parameter λ would be interpreted as the fluidity of the medium in which the swarm moves. This may serve as an explanation to why some research on the subject has encouraged the use of a large inertia weight at first and then to gradually reduce it to a much lower value. A high value yields extensive exploration by the swarm (which would correspond to a low viscosity medium in the terminology of fluid dynamics), and a low value to detailed exploitation (a dissipative medium). A popular approach is to let the inertia weight decay linearly from 0.9 to 0.4 during the search. There have however been numerous propositions in the literature. Eberhart and Shi proposed an adaptation of the inertia weight using a fuzzy system ([23]), i.e. a system that analyzes analog input in terms of logical variables that take on continuous values in the range $[0, 1]$, which is in contrast to digital logic. This approach is more complicated to implement and analyze than using a linearly decaying inertia weight, but studies have shown that it can significantly improve the performance of PSO. Another method that has useful is to use an inertia weight with a random component, rather than time-decaying. For instance, Eberhart and Shi achieved good results using an inertia weight that was drawn from a uniform distribution in the range $[0.5, 1]$ at each iteration ([24]). Successful experiments have even been carried out by Zheng, Ma, Zhang and Qian using increasing inertia weights ([25]).

4.2 Handling constraints

Since many algorithms in soft projected clustering use objective functions with several constraints, it may be of interest to see how these issues can be resolved in PSO.

The most straight-forward way of dealing with constraints is to always preserve feasibility of the solutions. Any update that would cause a particle to fly outside the feasible space is discarded, and the particle is left unchanged until the next iteration. In this way, the swarm searches the whole space but only keeps tracking feasible solutions. To accelerate this process, all the particles are randomly initialized in the feasible space. The major drawback of this approach is that every time an update is discarded, an iteration for that particle is "lost", which can slow down convergence. It is on the other hand easily implemented and may be utilized for any kind of constraint.

There are however several methods, more or less fruitful, that deal with constraints in PSO. Koziel and Michalewicz divided these into four categories ([26]): methods based on preserving feasibility of solutions, methods based on penalty functions, methods that make a clear distinction between feasible and infeasible solutions, and other hybrid methods. The principles of the first category have already been explained. The second widely used group are based on penalty functions. These methods use penalty functions to penalize particles that are outside the feasible space, by giving them worse fitness values than particles that are feasible. In this way, infeasible solutions are encouraged to fly towards the feasible space. Although penalty functions are easily implemented, the penalty factors that determine the impact of the penalty function are difficult to set to a suitable value. One way of getting around this difficulty is to use a self-adapting scheme instead. He and Wang did just that ([27]). They used two swarms in a co-evolutionary fashion; one swarm kept track of self-adapting penalty factors and the other was used in parallel to find good decision solutions. Ray and Liew extend the use of penalty functions by introducing a constraint matrix with one entry for each particle on each constraint ([28]). The constraint matrix is then updated according to the penalty functions to find good solutions in the feasible space.

4.3 The CLPSO variant

The Comprehensive Learning Particle Swarm Optimizer (CLPSO) was proposed by Liang, Qin, Suganthan and Baskar in 2006 ([29]), and has recently risen to become one of the most well-known modifications of PSO. The motivation behind the algorithm was to come to terms with the problem of premature convergence in the original PSO. It has been found that PSO may easily get trapped in a local optimum when solving complex multi-

modal problems where several local optima arise. In the original version of PSO, each particle learns from its personal best and the swarm's global best in parallel. Restricting the social learning ability to only the global best makes the original PSO converge fast. On the down-side, because all particles in the swarm learn from the global best even if the current global best happens to be far from the global optimum, the swarm may easily be attracted to a region containing the global best and get trapped in a local optimum if the search environment is complex enough. CLPSO employs a different learning strategy to avoid this, where all particles' historical best positions are used to update the particle velocities. This approach preserves the diversity of the swarm to prevent premature convergence.

In CLPSO, the particles are updated according to

$$v_i(t+1) = \lambda v_i(t) + \phi r \left[c_i(t) - x_i(t)\right], \tag{4.3}$$
$$x_i(t+1) = x_i(t) + v_i(t+1), \tag{4.4}$$

where the parameter ϕ is an acceleration factor and r is a random number uniformly distributed in the range $[0, 1]$ that is updated at each iteration.

4.3.1 Crossover learning

Notably, there is a different information-sharing strategy in CLPSO than in classic PSO. In the particle velocity update above, c_i is a comprehensive learning result for particle i (which is sometimes denoted *Cpbest* in the literature). It is produced from the personal best position of the particle itself and one of the other best personal positions in the swarm according to a crossover operation such that the particle learns from a good exemplar.

The update of a comprehensive best largely depends on the probability P_c, called the *learning probability*, which can take different values for different particles. For each dimension of a particle, a random number in between zero and one is generated by the algorithm. If the random number happens to be larger than the P_c value for that particle, the corresponding dimension will learn solely from its own personal best, otherwise it will learn from another particle's personal best. The other personal best is obtained through a tournament mechanism which operates as follows:

- Firstly, two particles of the swarm are chosen randomly which excludes the particle whose velocity is being updated.
- The fitness values of the selected particles are then compared, and only the better one is considered onwards (depending on whether a minimization or a maximization problem is considered).
- The winner is then used to learn from for that dimension. If all exemplars of a particle happens to be its own, a dimension is randomly

chosen to learn from another particle's corresponding dimension. In this way we can always guarantee a social behaviour of each particle.

All the comprehensive bests have the ability to generate new positions in the feasible space using information provided by different particles' history. Since each dimension is treated separately, the comprehensive best is rarely influenced by only one particle, in contrast to PSO where the particle is influenced only by its personal best and the swarm global best. To further ensure that a particle learns from good exemplars in CLPSO, and also to avoid searching along poor directions, the particle is allowed to learn from the exemplars until the particle no longer improves for a certain number of iterations. This threshold is called the *refreshing gap*, and whenever it is reached for a particle, the comprehensive best for the particle is reassigned. The tuning of the refreshing gap is problem-specific, but a value in between 5 and 10 can be recommended for most applications.

The CLPSO algorithm for a minimization problem is summarized as follows, given the objective function f (the maximization problem is treated analogously):

Initialization:

Randomly initialize the position and velocity swarms, X and V.

Initialize the personal best positions by setting $p_i = x_i$, for each particle in X.

Store the swarm of personal bests as P.

Evaluate the fitness of each particle in X by the objective function f.

Initialize the global best position by setting g to the best particle in P:

$$g = \arg\min_i f(p_i)$$

Repeat:

Produce the comprehensive best positions c_i from P, for each particle in X.

Update for each particle in V according to (4.3).

Update for each particle in X according to (4.4).

Update for each particle in P according to

$$p_i = \begin{cases} x_i, & \text{if } f(x_i) < f(p_i) \\ p_i, & \text{otherwise} \end{cases}$$

Update the global best according to

$$g = \arg\min_i f(p_i)$$

Until: (the objective function reaches a global minimum value, or the number of function evaluations reaches a specified threshold).

Notably, there are essentially three aspects where CLPSO is different from classic PSO:

- The information-sharing strategy: instead of using a particle's own personal best and the swarm global best as influences, all particles' personal bests can potentially be used as exemplars to learn from.
- Instead of learning from the same exemplar particle for all dimensions, each dimension of a particle in general learns from different personal bests for different dimensions during a few iterations. In other words, each dimension of a particle may learn from the corresponding dimension of different particles' personal bests.
- Contrary to PSO, where the particles learn from two exemplars (personal and global best) at the same time in every iteration, each dimension of a particle learns from just one exemplar during a few iterations in CLPSO.

4.4 Related methods

Apart from PSO and its variants, there have been many other computational intelligence-based algorithm proposed to solve the variable weighting problem in soft projected clustering. Two of the most widely used are the genetic algorithm (GA) and the ant colony optimization (ACO) technique.

GA is a stochastic search procedure based on the dynamics of natural selection, genetics and evolution. In GA, problems are thus solved by simulating processes that can be seen in nature, similar to PSO. Based on Darwin's principle of survival of the fittest, GA iteratively finds new and better solutions with few assumptions on the objective function. The idea is to keep a population of candidate solutions, each of which are within the feasible space. Each solution is in general coded as a binary string called chromosome. When the chromosome has been decoded, its fitness is evaluated using the objective function. Prominent chromosomes with better fitness values than others then go through a series of genetic operations such as crossover and mutation, like in nature, to form a new population. This procedure is repeated and evolves towards better solutions over generations until a satisfactory solution is obtained.

Although GA has been to converge to global optima when applied to several common test functions, there are a few drawbacks. One problem is that there are many internal parameters that have to be set for each problem, in contrast to PSO which only has a few. Tuning might be very time-consuming but is essential for obtaining good results. Another drawback is the huge number of fitness evaluations required by the algorithm, due to its relatively poor local search capability. In between 50 000 and 100 000 evaluations is not uncommon for normal usage, which is a considerable workload. Population diversity is also often a critical issue in GA. If the population is not diverse enough, it may cause repeated search and even lead to premature convergence.

ACO is a swarm intelligence technique that was introduced a few years before PSO. In nature, ants initially wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants happen to traverse the trail, they are likely to start following it, instead of continuing their random walk, returning and reinforcing it if they eventually find food. Based on this principle, it is essentially a probabilistic method for solving problems arising in graph theory, but can be applied equally well to optimization problems. For instance, it has been used to find near-optimal solutions to the traveling salesman problem. In this interpretation, ants correspond to feasible solutions of the shortest path problem. The more one certain possible solution is chosen the more likely it gets to be the optimum solution. Every solution for a given path happens with a certain probability which depends on the pheromones. The probabilities can be thought of as measures of how attractive a path is. The more one path is chosen the more attractive it becomes for other ants.

All computational intelligence-based techniques, such as the above, are suitable for several optimization problems where other methods fail to converge. Beside yielding good convergence in many cases, they often do not require much from the objective function, such as continuity or unimodality. Hence these methods have an advantage over traditional gradient-based approaches and can even perform well on black-box optimization problems (where the objective function is not known explicitly). They can thus be effectively applied to nonlinear optimization problems. PSO is no exception, and has capable of generating high-quality solutions within an acceptable computational cost and stable convergence characteristics. It is therefore a strong competitor to both GA and ACO and other intelligence-based techniques for solving the variable weighting problem for high-dimensional clustering. A few of its advantages are:

- PSO is mathematically tractable and easier to implement. PSO only needs two simple arithmetic operations (addition and multiplication), while GA requires implementations of much more complicated operators for dealing with selection and mutation. PSO is also more

computationally efficient than both GA and ACO. There are fewer user-dependent parameters to adjust.

- PSO has an intelligent information-sharing mechanism. Every particle remembers its own historic best position, whereas every ant in ACO needs to track down a series of its own previous positions and individuals in GA have no memory at all. As a result, a particle in PSO requires less time to calculate its fitness value than an ant in ACO.

- PSO is better suited for preserving swarm diversity and consequently better at avoiding premature convergence. Because of its information-sharing mechanism, the particles fly in the feasible space using their own previous best position as well as the swarm's previous best position. In GA, there is no cooperation, only survival of the fittest according to natural selection; the worst individuals are rejected and only the good ones survive. Neither ACO has direct cooperation between individuals and it may easily lose population diversity because ants are attracted by the largest pheromone trail.

- PSO has proven robust in many settings, especially in solving continuous nonlinear optimization problems, whereas GA and ACO are preferred for constrained discrete optimization problems. ACO has an advantage over genetic algorithm approaches and other evolutionary methods when the graph may change dynamically, because the ant colony algorithm can be run continuously and adapt to changes in real time.

5 The PSOVW algorithm

It is now time to present the PSOVW algorithm in detail. As mentioned, it was introduced by Lu et al. in 2009 to solve the variable weighting problem in clustering high-dimensional data ([12]). In PSOVW, the following objective function is employed:

$$F(W) = \sum_{l=1}^{k} \sum_{i=1}^{n} \sum_{j=1}^{m} u_{l,i} \cdot \left(\frac{w_{l,j}}{\sum_{s=1}^{m} w_{l,s}} \right)^{\beta} \cdot d(x_{i,j}, z_{l,j}) \tag{5.1}$$

subject to the constraints

$$\begin{cases} 0 \leq w_{l,j} \leq 1, & 1 \leq l \leq k, 1 \leq j \leq m \\ \sum_{l=1}^{k} u_{l,i} = 1, & u_{l,i} \in \{0,1\}, 1 \leq i \leq n. \end{cases} \tag{5.2}$$

Notably, the constraints are simpler than in most other algorithms arising in soft projected clustering applications, such as the LAC, the EWKM and the W-k-means algorithms. These three all include not only bound constraints but also equality constraints involving the variable weights. In PSOVW, these equality constraints are omitted by using a normalized representation of the variable weights in the objective function instead, which is one major advantage of PSOVW.

Another advantage of PSOVW is its search strategy, which is based on CLPSO. As previously stated, any PSO variant is well-suited for optimizing nonlinear functions with discontinuous gradients, with gradients that are hard to calculate explicitly or in any other situation where gradient-based methods are not applicable. It should also be noted that PSO requires that the equality constraints are eliminated. Otherwise, every particle would have to keep track of the other particles, such that no bounds would be exceeded. With only bound constraints, the particles need to be responsible only for themselves, making sure that the bound constraints are not violated on their part.

The objective function in PSOVW is actually a generalization of a collection of objective functions already employed in soft projected clustering. If $\beta = 0$, function (5.1) shares a strong resemblance to the objective function used in the k-means algorithm. In fact, the only differences between them are the representation of variable weights and the constraints. If $\beta = 1$, function (5.1) is also similar in the same way to the objective function in EWKM, except for the terms including weight entropy. If we drop the second index on the variable weights, that is, $w_{l,j} = w_j$, then (5.1) is similar to the objective function in the W-k-means algorithm, which assigns a single variable weight vector. The difference is once again in the representation of variable weights and the constraints.

In PSOVW, five swarms are kept:

- The position swarm W of variable weights
- The velocity swarm V
- The swarm Z of cluster centroids
- The swarm of personal bests P
- The swarm of comprehensive bests C

In each of the swarms, an individual is represented by a $k \times m$ matrix. Apart from these swarms, the algorithm keeps track of the cluster memberships (represented by a vector of size n) and the global best position found so far (also represented by a $k \times m$ matrix).

At the first stage of the algorithm, the position swarm W, the velocity swarm V and the swarm of cluster centroids Z are chosen randomly. Then, given the variable weight matrix and the cluster centroids, the cluster membership of each data object is determined by the following formula:

$$\begin{cases} u_{l,i} = 1, & \text{if } \sum_{j=1}^{m} \left(\frac{w_{l,j}}{\sum_{s=1}^{m} w_{l,s}} \right)^{\beta} \cdot d\left(z_{l,j}, x_{i,j}\right) \leq \sum_{j=1}^{m} \left(\frac{w_{q,j}}{\sum_{s=1}^{m} w_{q,s}} \right)^{\beta} \cdot d\left(z_{q,j}, x_{i,j}\right), \\ & \text{for } 1 \leq q \leq k, \\ u_{l,i} = 0, & \text{for } q \neq l \end{cases} \tag{5.3}$$

Once the cluster membership is obtained, the cluster centroids are calculated by

$$z_{l,j} = \left(\sum_{i=1}^{n} u_{l,i} \cdot x_{i,j} \right) \Big/ \left(\sum_{i=1}^{n} u_{l,i} \right), \quad \text{for } 1 \leq l \leq k, 1 \leq j \leq m.$$

In this formula, the denominator is the number of objects in the cluster l resulting from the membership update in (5.3). The numerator is the sum of the values of the data objects in the cluster along dimension j. Hence, each dimension of a centroid is updated by the mean of the values of the data objects on that dimension. This is a straight-forward and intuitive approach, as it centralizes the cluster centroids among the objects in the cluster. The same strategy is employed in the LAC, the W-k-means and the EWKM algorithms. Should an empty cluster result from the membership update by (5.3), the PSOVW algorithm will randomly select a data object out of the data set to reinitialize the cluster centroid.

The PSOVW algorithm can be summarized as follows:

Initialization:

Randomly initialize the velocity swarm V in the range $[-v_{\max}, v_{\max}]$, where $v_{\max}$ is the velocity clamping threshold.

Randomly initialize the position swarm W in the range $[0, 1]$.

Randomly initialize the swarm Z of cluster centroids by selecting a set of k data objects out of the data set.

For each position in W, evaluate its fitness by (5.1).

Initialize the swarm of personal bests P.

Initialize the swarm of comprehensive bests C from P.

Initialize the global best position g.

Repeat:

Update the swarm of comprehensive bests C from P.

Update the velocity swarm V by (4.3).

Update the position swarm W by (4.4).

For each position in W,

If W lies in the range $[0, 1]$,

Evaluate its fitness by (5.1).

Update the position's personal best and store it in P.

Otherwise,

W is neglected.

End

If P has changed, update the global best g.

Until: (the objective function reaches a global minimum value, or the number of function evaluations reaches a specified threshold).

Post-processing: (partition the data objects by formula (5.3) with the weights of the global best position).

5.1 Computational complexity

Let s be the swarm size utilized in the algorithm, i.e. the number of particles in the swarm, and let T be the number of iterations needed for convergence. Then the runtime complexity of PSOVW can be analyzed as follows. If we assume that the effects of the initialization and the post-processing are negligible, then we can focus only on the main loop. During an iteration in the main loop, the following is performed:

- **Particle updates.** The comprehensive best, the velocity and the position is firstly updated for each particle. Since these three are all represented by $k \times m$ matrices, this procedure needs $O(mk)$ operations for each particle, so the complexity is $O(smk)$ in total.
- **Fitness evaluations.** Given the weight matrix W and the cluster centroids matrix Z, each particle's fitness value is evaluated. During this step, the data objects are partitioned and then new clusters centroids are determined. The cluster membership update requires $O(mnk)$ operations for each particle, which adds up to a total cost of $O(smnk)$ operations. The complexity for assigning new cluster centroids is $O(mk)$ for each particle. In total, the procedure of evaluating all particles fitness values needs $O(smnk)$ operations.
- **Personal best updates.** In the last step, each particle's personal best position is updated. This operation requires $O(mk)$ operations for each particle, since each personal best particle is stored as an $k \times m$ matrix. This yields a complexity of $O(smk)$ in total.

Consequently, if T is the number of iterations performed, then the total computational complexity is $O(smnkT)$. Hence, the PSOVW algorithm is scalable to the swarm size, the number of dimensions, the number of data objects and the number of clusters. The corresponding computational cost for the LAC, the W-k-means and the EWKM algorithm is $O(mnkT)$, so their computational complexity is lower and increases linearly as the number of dimensions, the number of data objects or the number of clusters increases.

5.2 Performance

Although PSOVW needs more resources than the other algorithms, it has been suggested that the extra computational time is acceptable in relation to the clustering results achieved. In [12] for instance, Lu et al. performed an extensive comparison between the four methods, using several simulated data sets. Their conclusion was that PSOVW outperformed the other algorithms in terms of clustering accuracy and clustering variance on most data

sets, at a cost of longer running time, but at a running time that was found acceptable.

Three main reasons behind PSOVW's superior performance were proposed:

- Foremost, PSOVW has a much more complicated and efficient search strategy than its contestants. Because it employs the PSO approach to the clustering problems, the particles do not work independently but interact with each other to move to better regions, in contrast to the other algorithms.
- PSOVW combines a k-means like function with a normalized representation of variable weights to form an objective function that is subject only to bound constraints. The other algorithms are subject to both bound constraints as well as equality constraints.
- PSOVW is less sensitive to the initial cluster centroids than the other algorithms. This resulted in the least variance in clustering accuracy in most of the data sets during the numerical experiments.

It should however be pointed out that PSOVW requires more parameters to be set by the user than other k-means like algorithms. The PSOVW is consequently more reliant on the user's parameter choices compared to similar algorithms in this field. When it comes to PSOVW, good results are often synonymous with good parameter choices.

6 The UPSOVW algorithm

Having introduced and presented the PSOVW algorithm in detail, we are now ready to fully analyze the algorithm and to propose a few potential improvements of it.

As mentioned, any successful algorithm in soft projected clustering is largely dependent on its objective function and its search strategy. In PSOVW, the choice of objective function is partly motivated by the desire to eliminate the equality constraints on the variable weights. These can be omitted by using a normalized representation of variable weights, leaving only bound constraints. The bound constraints are then handled by preserving feasibility of the solutions, which is the natural way of dealing with this type of constraint. Any update that would cause a dimension to exceed a bound is discarded, which guarantees feasible solutions at any stage of the algorithm. The major drawback of this approach is that every time an update is discarded, an iteration for that dimension is "lost", which affects the corresponding particle's trajectory and may result in slower convergence.

Instead, we propose a different approach for getting around this problem without diminishing the search capabilities. By introducing a suitable scaling of the objective function used in PSOVW, we can guarantee that the resulting variable weights are in the feasible space. Before describing the scaling in detail, we introduce the modified objective function

$$G(W) = \sum_{l=1}^{k} \sum_{i=1}^{n} \sum_{j=1}^{m} u_{l,i} \cdot \left(\frac{|w_{l,j}|}{\sum_{j=1}^{m} |w_{l,j}|} \right)^{\beta} \cdot d(x_{i,j}, z_{l,j}),$$

which is essentially the same objective function F used in PSOVW, except that the absolute values of the variable weights are considered. This is in effect the same as applying F to a variable weight matrix with nonnegative entries. So, the design of G implies that the resulting variable weights meet the lower bound of the bound constraints in (5.2). It remains to deal with the upper bound of the constraints. One can show that F, as well as G, is unaffected by scaling, that is $F(\alpha W) = F(W)$, for all scalars α. Hence, if there is a scalar such that the resulting variable weights are within the limits, we can drop the bound constraints altogether. The next theorem provides the details on how such a scalar may be obtained and how this strategy can be used to transform the optimization problem considered in PSOVW into an equivalent one without bound constraints.

Main theorem. *Consider the optimization problem*

$$\begin{aligned} & \min G(W) \\ \textit{subject to} \quad & \sum_{l=1}^{k} u_{l,i} = 1, \quad u_{l,i} \in \{0,1\}, 1 \leq i \leq n. \end{aligned} \tag{6.1}$$

Suppose that $W^* \in \mathbf{R}^{k\times m}$ *is a minimizer of* G *subject to* (6.1)*. Further, let* $w^*_{\max}$ *be the entry of maximum magnitude of this matrix, that is,*

$$w^*_{\max} = \max\left\{|w^*_{l,j}| \colon 1 \leq l \leq k, 1 \leq j \leq m\right\}.$$

Define a new matrix $W\prime$ *with entries*

$$w_{l,j}\prime = \frac{\left|w^*_{l,j}\right|}{w^*_{\max}}, \quad 1 \leq l \leq k, 1 \leq j \leq m.$$

Then $W\prime$ *is a minimizer of* F *and meets the bound constraints in* (5.2) *used in PSOVW.*

Although a seemingly simple observation, the theorem gives an idea of how a PSO method (or in fact any optimization method we may see fit) can be used without having to deal with bound constraints on the variable weights. The theorem essentially states that we may equally well solve the optimization problem involving G with fewer constraints than to solve the more complicated problem involving F. Beside the objective function, the variable weights have an impact on the membership update (5.3) in PSOVW. However, by the same reasoning as in the proof above, we can show that the clustering is unaffected by scaling of the variable weights.

The theorem is the main theorem in the thesis, simply because it will serve as the foundation of our proposed algorithm for soft projected clustering of high dimensional data. We call it *Unconstrained Particle Swarm Optimizer for Variable Weighting* (UPSOVW)[1], since it is reminiscent of PSOVW but employs an unconstrained search strategy. In UPSOVW, we will stick to the CLPSO method for two reasons; firstly because it is an efficient optimization method for this kind of problem, and secondly because we would like to measure the impact of our new approach attributed to the main theorem.

The UPSOVW algorithm can be summarized as follows:

Initialization:

Randomly initialize the velocity swarm V in the range $[-v_{\max}, v_{\max}]$, where $v_{\max}$ is the velocity clamping threshold used in PSOVW.

Randomly initialize the position swarm W in the range $[0, 1]$.

Randomly initialize the swarm Z of cluster centroids by selecting a set of k data objects out of the data set.

For each position in W, evaluate its fitness by (5.1).

[1]Not to be confused with the *Unified Particle Swarm Optimizer* (UPSO), which is a variant of PSO. See [30].

Initialize the swarm of personal bests P.

Initialize the swarm of comprehensive bests C from P.

Initialize the global best position g.

Repeat:

Update the swarm of comprehensive bests C from P.

Update the velocity swarm V by (4.3).

Update the position swarm W by (4.4).

For each position in W,

Evaluate its fitness by (5.1).

Update the position's personal best and store it in P.

If P has changed, update the global best g.

Until: (the objective function reaches a global minimum value, or the number of function evaluations reaches a specified threshold).

Post-processing: (partition the data objects by formula (5.3) with the weights of the global best position).

Notably, the UPSOVW algorithm behaves similarly to PSOVW, except for in a few crucial respects. First and foremost, the particles are never deemed infeasible solutions, so every update on every particle of each iteration contributes to the overall convergence of the algorithm. Although the bound constraints could have been handled with one the methods explained in section 4.2 instead, for instance by using penalty functions, the theorem actually makes it possible to get rid of their effect altogether. However, it should be noted that all particles are initialized in the range $[0, 1]$, i.e. within the feasible space of PSOVW, which is mostly conventional. The second difference between UPSOVW and PSOVW is that velocity clamping is not employed in the former. However, the individuals in the velocity swarm are initially set within the range of the velocity clamping threshold, i.e. in $[-v_{\max}, v_{\max}]$. In UPSOVW, the threshold value is also chosen by convention, since it has a small impact on the final result of the algorithm, in contrast to PSOVW where it plays a much stronger role. Otherwise, the guidelines for tuning the parameters in UPSOVW should follow those of PSOVW. The computational complexity of the algorithms are the same, although one may suspect that PSOVW needs more iterations to converge to the global optimum.

7 Theoretical analysis

We have already argued that the major advantage of UPSOVW lies in its unrestrained search capability based on the CLPSO method. However, because neither the particles' positions nor the corresponding velocities are bounded on their magnitudes, contrary to PSOVW, the immediate question of stability arises. Does UPSOVW yield a stable process as iterations go by? Even so, can convergence to the global optimum be guaranteed? Needless to say, a thorough analysis is in place to further motivate the use of UPSOVW and to analytically examine how its internal parameters should be set to ensure stability and at the same time sustain high performance of the algorithm.

Although there has been a considerable amount of research on the analysis of particle swarm optimization techniques, there is to date no coherent analysis that focuses exclusively on the CLPSO method. Despite this, most results attributed to classic PSO will be of interest in the study of UPSOVW as well, since CLPSO is essentially a particle swarm optimization technique. In other cases, the methods need only slight adjustments to fit into the framework of CLPSO.

7.1 Previous work

The PSO method may seem simple at first sight; it is easy to implement and requires only a few of the arithmetic operators. However, viewed from a strictly theoretical aspect, the PSO offers several challenging problems of interest to anyone eager to understand swarm intelligence through theoretical analysis. In fact, there is to date no extensive mathematical model of particle swarm optimization that encapsulates all theoretical aspects. There are numerous reasons for this.

First and foremost, the particle swarm technique is founded on a large number of cooperating individuals (the particles). Although the principles of a single individual's movement and the rules of its interaction with other particles are trivial, the behaviour of the whole swarm is not. The difficulties in modeling the swarm dynamics can largely be attributed to the particles' self-awareness. Since each particle is provided with memory and the ability to decide when to update the memory, each particle is capable of changing its current direction from one iteration to the next. Secondly, the forces acting on the particles are stochastic. Hence, any attempt to analyze the stability issues should avoid classical analysis methods used in dynamical systems, such as eigenvalue analysis. Thirdly, since PSO is an optimization tool, the objective function in consideration naturally plays a strong role in the performance of the algorithm. There are however infinitely many objective functions and it is consequently extremely difficult to draw any general conclusions that apply to all of them. In fact, this is difficult even

in special cases, when the objective function is known explicitly. Because the particles are randomly scattered in the search space during initialization and the particles are also influenced by stochastic forces, infinitely many scenarios can potentially be created because of the stochastic nature of the algorithm.

Despite these issues, several useful contributions have been presented in this fields in recent years. Some of them will be summarized in this section.

7.1.1 Deterministic models

Many attempts to fully understand the subtleties of the PSO method from a theoretical viewpoint have failed, and several researchers in the PSO community have had to revert to the study of more mathematically tractable models instead. These models are often based on simplifying assumptions. For instance, models that only consider single isolated particles are common. Models that focus exclusively on convergence during stagnation (when no improved solutions are found) and models that disregard the stochastic aspect altogether are other popular approaches. Some models combine several assumptions to simplify the analysis. Although the simplifications may illustrate some of the features of the algorithm more clearly, their effect on the swarm dynamics may interfere with the true dynamics of the algorithm. Consequently, any results obtained from such an observation should be treated as approximations and should be determined empirically with the original system.

The first analysis of a deterministic model worthy of recognition was carried out by Ozcan and Mohan a few years after the advent of PSO ([31]). They studied a simplified model with a swarm consisting of a single particle, in isolation, in one dimension and during stagnation. Also, no inertia weight nor velocity clamping was considered, and every effect of randomness was discarded. In their following work on the subject, many of these assumptions were dropped, and the model was instead extended to hold for swarms containing several multi-dimensional particles ([32]). In this setting, they were able to derive solution trajectories for the particles. They found that, in search for a global optimum, each particle in the system randomly chooses a path on a sinusoidal wave by altering the wave's frequency and amplitude. Consequently, a particle does not "fly" in the search space, but rather "surfs" it on these sine waves, and the phenomenon is therefore called "surfing the waves". Their second discovery showed an interesting coupling between the acceleration factors, ϕ_p and ϕ_g. Apparently, the behaviour of PSO depends on the value of the sum $\phi_p + \phi_g$. For instance, the model proposed that in case $\phi_p + \phi_g > 4$, the particles will oscillate with increasing amplitudes, which may eventually lead to an explosion of the system. Conversely, for $\phi_p + \phi_g < 4$, the model suggested that particles follow bounded periodic oscillations, yielding stable dynamics.

Similar assumptions were used in a model developed by Clerc and Kennedy a few years later ([33]). They considered a single particle of one dimension, under no stochastic influence and during stagnation. The propagation of the resulting swarm could then be expressed as a linear dynamical system in discrete-time. The dynamics of the state (position and velocity) of the particle could then be computed by finding the eigenvalues and the corresponding eigenvectors of the state transition matrix. Depending on the eigenvalues, it was suggested that the particle would converge to an equilibrium point; a weighted average of the particle's best position found so far and the global best, the acceleration factors acting as the weights. According to classical systems theory, the particle will converge to equilibrium if the eigenvalues lie within the unit circle. Since the eigenvalues of the system are essentially dependent on the internal parameters in PSO, the model could provide some guidance of how parameters should be chosen so as to achieve convergence.

Their study is summarized in more detail below, although the effect of the inertia weight is also studied in the following. If the swarm is in a stagnation phase, with no fitness improvements, each particle effectively acts independently. So, it is sufficient to observe a single particle in this setting. Thus, we may drop the indices, and express the equations describing the particle's motion as

$$\begin{aligned} v(t+1) &= \lambda v(t) + \omega_p(t)\left[p(t) - x(t)\right] + \omega_g(t)\left[g(t) - x(t)\right], \\ x(t+1) &= x(t) + v(t+1). \end{aligned}$$

This is a more compact way of describing the dynamics of the particle. The parameters ω_p and ω_g are in this interpretation the stochastic forces acting on the system. They are random numbers uniformly distributed in the range $[0, \phi_p]$ and $[0, \phi_g]$, respectively, where ϕ_p and ϕ_g are the acceleration factors. It should be noted that everything except for the inertia weight is considered to be time-varying, although the analysis may be extended to hold for time-varying inertia weights as well. As a next step, we may introduce the state vector by $\xi(t) = [v(t)\ \ x(t)]^T$. Moreover, under the assumption that the stochastic forces are in fact constant, that is $\omega_p(t) = \phi_p$ and $\omega_g(t) = \phi_g$, we can formulate the particle dynamics as the following dynamical system:

$$\xi(t+1) = \begin{bmatrix} \lambda & -(\phi_p + \phi_g) \\ \lambda & 1 - (\phi_p + \phi_g) \end{bmatrix} \xi(t) + \begin{bmatrix} \phi_p & \phi_g \\ \phi_p & \phi_g \end{bmatrix} \begin{bmatrix} p(t) \\ g(t) \end{bmatrix}$$

This system is linear time-invariant (LTI) with exogenous inputs. Although the exogenous input vector containing p and g generally depends on the state vector in PSO, we can neglect the effect of this dependence if stagnation is assumed, i.e. if the exogenous input remains constant. From standard results in control theory follows that the stability of the system depends on

the eigenvalues of the system matrix. The eigenvalues r_1 and r_2 are the solutions to the characteristic equation

$$r^2 + (\phi_p + \phi_g - \lambda - 1)\, r + \lambda = 0.$$

The necessary and sufficient condition for stability of the system is that both of these eigenvalues (whether real or complex) lie within the unit circle of the complex plane. This condition can easily be verified without computing the eigenvalues explicitly. A result derived from the Routh-Hurwitz theorem states that the equation $z^2 + az + b = 0$ has its roots within the unit circle if

$$\begin{aligned} |b| &< 1, \\ 1 + a + b &> 0, \\ 1 - a + b &> 0. \end{aligned}$$

In our case, the condition for stability thus becomes

$$\begin{aligned} -1 &< \lambda < 1, \\ 0 &< \phi_p + \phi_g < 2(1 + \lambda). \end{aligned}$$

The parameter region giving stability is presented as the triangular area in figure 1. Interestingly enough, the same stability region is obtained if this simplified model is applied to CLPSO instead. Notably, the eigenvalue analysis holds for the corresponding CLPSO system by setting $\phi = \phi_p + \phi_g$. Although the analysis is inaccurate due to the assumptions it relies on, it clearly shows that the inertia weight and the acceleration factors cannot be set in isolation. Because the stability effectively depends on the sum of the acceleration factors, and not on the bounds of ϕ_p and ϕ_g individually, the analysis suggests that there are opportunities to control the level of exploration and exploitation while maintaining stability.

Campana, Fasano, Peri and Pinto reused and extended the view of the population dynamics as a dynamical system ([34]). In addition, they also included the inertia weight in their model, which Clerc and Kennedy effectively had omitted in [33]. On the condition that no random behaviour is taken into account, the corresponding model turns into a discrete-time, linear and stationary system. Thus, the system's state space trajectories can be expressed as the sum of two separate trajectories: the *free response* and the *forced response*. Because the system at hand is discrete-time, these can both be expressed explicitly by solving the recursive relation given by the system. They observed that the free response is independent of the personal bests and the global best, but uniquely dependent on the initial state of the system. Conversely, the forced response depends uniquely on the personal bests and the global best, and is independent of the initial state. However,

they were only able to give a detailed analysis of the free response, since the forced response is consequently dependent on the structure of the fitness function. Nevertheless, an eigenvalue analysis was carried out that illustrated the behaviours that can be expected from a particle given a set of parameter values. The analysis also suggested a few interesting guidelines on how the particles' positions and velocities should be initialized in order to obtain good results.

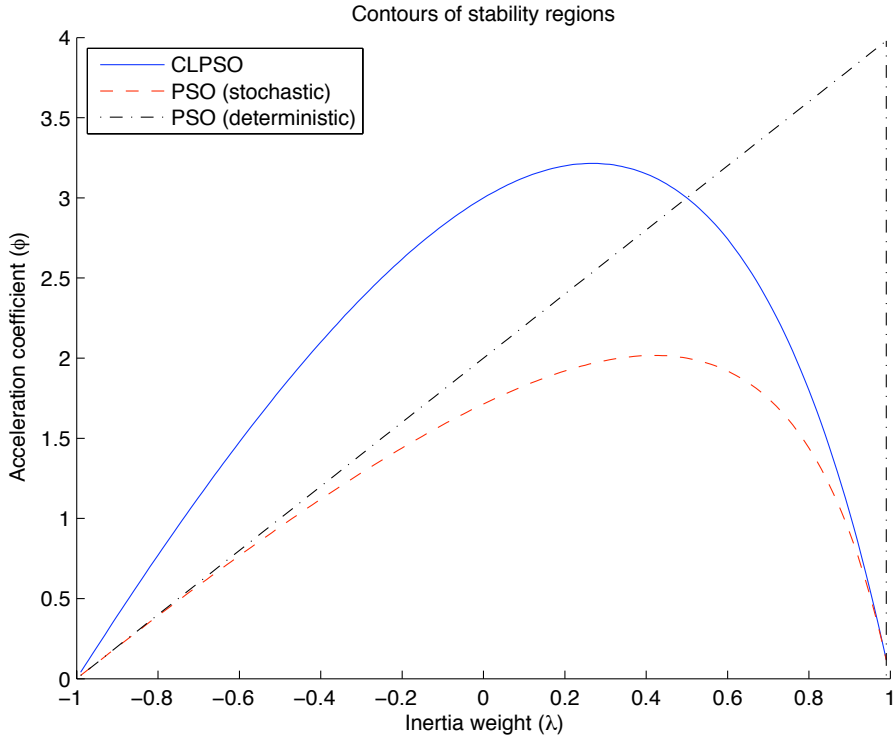

Figure 1: Parameter regions ensuring stability according to different methods of analysis.

7.1.2 Stochastic models

As have been seen, much of the research from the PSO community has, as a result of simplifying assumptions, evolved around deterministic systems, i.e. systems that do not account for the stochastic nature of swarm intelligence. Although this approach offers some tractable mathematical tools for analysis, it may also severely restrict the accuracy of the analysis, which makes a deterministic model hard to justify in this context. In PSO, there are stochastic forces acting on each particle, and the methods of analysis should be chosen accordingly. The rest of this section is therefore dedicated

to stochastic models for analysis of particle swarm optimization.

A pioneering work in this field was carried out only a year after the advent of PSO, in 1996, by Rudolph ([35]). His study was actually not exclusively aimed at swarm theory, but was intended to explain the consequences of randomness in any evolutionary algorithm. Instead, the evolutionary algorithm was viewed as a homogeneous Markov chain. In this setting, the population is modified by *genetic operators* that generate a new population at the next iteration. The resulting population only depends on the previous population and the probabilistic modifications caused by the genetic operators. Although this model is more suitable for evolutionary algorithms that already use genetic operators, such as the genetic algorithm, it could work equally well for PSO in principle. The necessary computations for such an application are however very complex in practice.

A similar approach was adopted by Poli, Langdon, Clerc and Stephens only a few years ago ([36]). They proposed a model based on discrete Markov chains that was designed to apply to arbitrary continuous problems. Their idea was to discretize the objective function using a finite element method (FEM) grid which would generate corresponding distinct states in the search algorithm. In contrast to the work of Rudolph, the discretization makes the computation of the transition matrix describing the system straight-forward. Empirical results confirmed that the predictions of the Markov chain were surprisingly accurate.

Poli also published a study on his own recently, but with a different aspect of the problem at hand ([37]). Under the sole assumption of stagnation, he studied the dynamics of the first and second moments of the sampling distribution in PSO, i.e. the mean and variance of the particle trajectories. Based on these results, the study revealed that the sampling distribution does not depend on where the personal bests and swarm best are located in the search space. An especially fundamental property of the stability was also presented; in order for a particle to converge, it is not sufficient to require that the mean of the the corresponding trajectory converges to an equilibrium point, but we must also have that the variance goes to zero. A few examples of converging trajectories were presented for some common parameter settings, but the parameter region ensuring stability was never determined analytically.

Jiang, Luoa and Yanga published a similar study ([38]), also involving the dynamics of the moments and providing a few guidelines on parameter selection. Kadirkamanathan, Selvarajah and Fleming, on the other hand, used Lyapunov theory and the concept of passive systems to prove stability in the presence of randomness ([39]). Their model included a single particle, the swarm best, with inertia weight and during stagnation. The particle trajectory was described by a dynamical system with nonlinear feedback control, and several useful results were found by means of control theory. For instance, they found the transfer function of the particle, proved ob-

servability and controllability of the system, found a quadratic Lyapunov function and found sufficient conditions on the PSO parameters that guarantee convergence.

7.2 A stochastic model with multiplicative noise

The UPSOVW algorithm is essentially based on conflicting interests; the aim is to let the swarm explore as much of the whole space as possible in quest for the global minimum, but at the same time make sure that the searching process is stable. The following paragraphs present a well-suited model applicable to the analysis of CLPSO that take this compromise in consideration. We also give a detailed study of the parameters therein, with an emphasis on parameter settings ensuring stability.

Any suitable model for analyzing swarm intelligence algorithms should be true to the real behaviour of the algorithms and at the same time be sufficiently tractable from a mathematical viewpoint. The CLPSO method is no exception to this rule of thumb. No matter how simplified a model may be, any model that aims at analyzing CLPSO is in general derived from the equations describing particle motion:

$$\begin{aligned} v_i(t+1) &= \lambda v_i(t) + \omega(t)\left[c_i(t) - x_i(t)\right], \\ x_i(t+1) &= x_i(t) + v_i(t+1). \end{aligned}$$

The parameter ω is as usual the stochastic force acting on the particle. It is a random number uniformly distributed in the range $[0, \phi]$, where ϕ is the acceleration factor. It should be noted that everything except for the inertia weight is considered to be time-varying, although the analysis may be extended to hold for time-varying inertia weights as well.

The study of such stochastic systems is central to the field of *robust control theory*. It aims at controlling uncertain dynamical systems such that the resulting system can tolerate variability and uncertainty. The uncertainties can be attributed to imperfect knowledge of some of the components of the system, or to changes in their behaviour due to altered operating conditions. Another cause stems from physical parameters whose values are only approximately known or time-varying. However, exogenous effects such as disturbances or measurement noise should not be considered as model uncertainty.

Because the swarm behaviour of CLPSO differs in some aspects to that of classic PSO, the assumptions that our model relies on should be adapted accordingly. During a swarm stagnation phase in PSO, neither the personal bests nor the global best are updated, so the particles can be considered to act independently in this setting. CLPSO, on the other hand, automatically changes the comprehensive best particles during stagnation, such that

each particle learns from a good exemplar. A particle's comprehensive best is updated only if the particle has not improved during a fixed number of iterations, which is set by the refreshing gap. Consequently, a particle is influenced by the same exemplar as long as it improves at some point before the refreshing gap threshold is encountered. We will thus not assume stagnation, which is common in classic PSO analysis, but instead consider a single particle that successively improves as iterations go by.

With this assumption, we may drop the indices in the equations to indicate that an arbitrary particle is observed, and let the comprehensive best be constant, $c = c(t)$. Moreover, to get rid of the exogenous component altogether, we introduce a new variable $y(t) = x(t) - c$, and can then express the particle's dynamics as the linear dynamical system

$$\xi(t+1) = \begin{bmatrix} \lambda & -\omega(t) \\ \lambda & 1-\omega(t) \end{bmatrix} \xi(t), \tag{7.1}$$

where ξ is the state vector, $\xi(t) = [v(t) \;\; y(t)]^T$. Because of its linearity, we can compute the equilibrium point of the system, ξ^*. The equilibrium point satisfies the equation

$$\xi^*(t+1) - \xi^*(t) = 0, \quad \text{for all } t \geq 0,$$

which is rearranged into a linear system yielding the unique (trivial) solution $\xi^* = [0 \;\; 0]^T$, implying that $v^* = 0$ and $x^* = c$ when the system is in equilibrium. This is quite a remarkable result since we are dealing with a stochastic system, whose parameters vary randomly from one iteration to the next.

Apart from this property, and the advantageous properties of linearity and the absence of exogenous inputs, the analysis is considerably simplified by the fact that velocity clamping is not included in UPSOVW. Since the velocities are unbounded on their magnitudes, they are completely determined by system (7.1), which guarantees high accuracy of the analysis. On the other hand, the fact that the system is time-varying makes the analysis significantly more complex. Unfortunately, an eigenvalue analysis similar to the above cannot be extended to these systems, so we must turn to other methods.

7.2.1 Multiplicative noise and mean-square stability

Recently, Wakasa, Tanaka and Akashi proposed an interesting aspect to the robust stability issues in particle swarm optimization ([40]). Instead of making unnecessary assumptions on the particle dynamics to obtain a more tractable problem, they considered a system similar to that of (7.1) and treated the stochastic components as multiplicative noise. Inspired by this

approach, we will derive a similar model that holds for CLPSO, but may easily be extended to hold for similar swarm intelligence algorithms.

We first consider the discrete-time stochastic system

$$\xi(t+1) = \Big(A_0 + A_p p(t)\Big)\xi(t), \tag{7.2}$$

where A_0 and A_1 are constant matrices of appropriate size and $p(0), p(1), \ldots,$ are independent, identically distributed (i.i.d.) random variables with mean and variance according to

$$\mathbf{E}\, p(t) = 0, \quad \mathbf{E}\, p(t)^2 = \sigma^2.$$

This is known as *single multiplicative noise*, since there is a single noisy parameter p in the model and the noise is propagated through multiplication (in contrast to additive noise) by a constant matrix.

Next, we define $M(t)$, the state correlation matrix at time k, as

$$M(t) = \mathbf{E}\, \xi(t)\xi(t)^T.$$

If we assume that the initial state $\xi_0 = \xi(0)$ is independent of the process p, it can then be shown by straight-forward computation that M satisfies the linear (deterministic) recursion

$$M(t+1) = A_0 M(t) A_0^T + \sigma^2 A_p M(t) A_p^T, \quad M(0) = \mathbf{E}\, \xi_0 \xi_0^T. \tag{7.3}$$

If this recursion is stable, for arbitrary choices of ξ_0, we say that the system is *mean-square stable*. The following definition provides further details about this type of stability.

Definition (Mean-square stability). The solution $\xi = 0$ of the stochastic system (7.2) is said to be *exponentially stable in the mean-square sense* if there exist constants $\alpha > 0$, $L > 0$ such that

$$\mathbf{E}\, \xi(t)\xi(t)^T \leq L\, e^{-\alpha t}\, \mathbf{E}\, \xi_0 \xi_0^T, \tag{7.4}$$

for any ξ_0 and any $t \geq 0$.

Mean-square stability is a strong form of stability. In particular, it implies stability of the mean $\mathbf{E}\, \xi(t)$ and that all trajectories converge to zero with probability one ([41], [42]). This approach is quite similar to that of Poli ([37]) in that the mean-square stability analysis also investigates the behaviour of the mean. However, there are two relevant linear systems in Poli; one for the mean and one for the variance of the trajectories. Once these are derived, the eigenvalues of the corresponding system matrices must be found to find necessary conditions for stability, which are both hard to compute in explicit form, let alone analyze.

Instead of solving the linear recursion (7.3) explicitly for the state correlation matrix M and perform an eigenvalue analysis, we will use Lyapunov methods to examine stability. By virtue of Lyapunov theory, it can be shown that mean-square stability is equivalent to the existence of a positive-definite matrix P satisfying the inequality

$$A_0^T P A_0 - P + \sigma^2 A_p^T P A_p < 0. \tag{7.5}$$

The inequality symbol in (7.5) means that the left-hand side must be negative-definite to satisfy the inequality, which is called a *linear matrix inequality* (LMI) since it is linear in the matrix variable P. The proof that (7.5) is in fact a sufficient and necessary condition can be carried out by applying a linear Lyapunov function of the form $V(M) = \text{trace}\,(MP)$. The details are found in appendix A.2. A brief introduction to LMIs is also included (see appendix B), which provides some basic definitions that are useful throughout this section as well as some fundamental properties aimed at the interested reader.

Before the Lyapunov method can be applied to the system of interest (7.1), it must be rewritten on the form of an affine system as of (7.2). By introducing a random variable $\theta(t)$ uniformly distributed in the range $[-1/2, 1/2]$, we see that

$$\mathbf{E}\,\theta(t) = 0, \quad \mathbf{E}\,\theta(t)^2 = \sigma^2 = \frac{1}{12},$$

and

$$\omega(t) = \phi \cdot \theta(t) + \frac{\phi}{2},$$

which ensures that ω is effectively uniformly distributed in the range $[0, \phi]$. The corresponding affine system is thus given by

$$\xi(t+1) = \Big(A_0 + A_\theta \theta(t)\Big)\xi(t),$$

where

$$\begin{aligned} A_0 &= \begin{bmatrix} \lambda & -\phi/2 \\ \lambda & 1-\phi/2 \end{bmatrix}, \\ A_\theta &= \begin{bmatrix} 0 & -\phi \\ 0 & -\phi \end{bmatrix}. \end{aligned}$$

7.2.2 Numerical results and stable parameter regions

What is left is thus the investigation of how the parameters λ and ϕ affect feasibility of the following LMIs:

$$\begin{aligned} P &> 0, \\ A_0^T P A_0 - P + \sigma^2 A_\theta^T P A_\theta &< 0. \end{aligned}$$

In particular, we are interest in finding a parameter region in the $\lambda\phi$-plane that ensures stability. However, feasibility (or infeasibility for that matter) of the LMIs are not easily proved analytically, even for fixed values of λ and ϕ. Fortunately, there are computational tools that can check feasibility and, if so, even calculate a Lyapunov matrix P that satisfies the LMIs. We have used the *LMI Control Toolbox* featured in MATLAB and in particular its LMI solver `feasp` to determine feasibility ([43]).

We have incorporated this functionality in a simple bisection algorithm to get an approximate parameter region for stability. Its objective is to maximize the parameter ϕ having λ given while maintaining a feasible solution to the LMIs. Since previous analyses have indicated that λ must have a magnitude less than one, values in this range will only be considered. The bisection algorithm can be summarized as follows:

- Produce a vector λ_{vec} of equidistant points in the range $[-1, 1]$.
- Set a threshold Δ for the absolute error of the computations.
- For each point λ in λ_{vec},

    Set upper and lower bound on ϕ,

        Let $\phi_{\text{low}} = 0$ and $\phi_{\text{high}} = 4$.

    Repeat while $|\phi_{\text{high}} - \phi_{\text{low}}| > \Delta$,

        Let $\phi = (\phi_{\text{high}} + \phi_{\text{low}})/2$.

        Compute the matrices A_0 and A_θ.

        Check feasibility of the corresponding LMIs.

        If feasible,

            Let $\phi_{\text{low}} = \phi$.

        Otherwise,

            Let $\phi_{\text{high}} = \phi$.

        End

    End

    Store ϕ_{low} together with λ.

End

The resulting parameter region ensuring stability can be viewed in figure 1. In the computations we have used $\Delta = 5 \cdot 10^{-4}$, so there is a guaranteed accuracy in 3 decimals.

With little effort, this approach can be extended to hold for the classic PSO method as well. The only difference is that there are two stochastic forces in this setting, ω_p and ω_g, and a different equilibrium point. However, if we assume that the acceleration factors coincide, that is, $\phi_p = \phi_g = \phi$, the resulting system can be expressed in terms of multiplicative noise, and can thus be treated in a similar manner to obtain a stable parameter region. The contours of this region are included in figure 1. Notably, under these assumptions, the parameter region of PSO is much smaller than that of CLPSO. For PSO, we may also conclude that the stochastic method with multiplicative noise is more restrictive than the deterministic method using eigenvalue analysis.

8 Synthetic data simulations

In this section a comprehensive study is conducted to evaluate the performance of the UPSOVW algorithm compared to that of PSOVW. The study is comprised of several scenarios, where the algorithms are run under the same conditions to ensure a fair comparison.

To conduct the comparisons, some underlying test data must be used, where the actual cluster structures are known beforehand. The algorithms' clustering results can then be compared to the intended clusters to determine the efficiency and accuracy of each algorithm. In real data sets there are often some missing data values, or they consist of a mix of discrete, continuous or even categorical and ordinal data. A few assumptions must inevitably be used to deal with these issues, which can be misleading or give erroneous results. For example, it is not obvious how a distance measure should be designed to cope with categorical or ordinal data values. The interested reader can find a variety of real data sets in the UCI database [2].

8.1 Generating synthetic data sets

Instead of tackling the issues that arise in the study of real data sets, we will instead use synthetic data sets that are randomly generated according to a set of predetermined rules. Synthetic test data is favourable in that it offers complete control of cluster overlapping and dimensionality. This is a crucial property since we are interested in how well and how fast each algorithm is able to retrieve known clusters in subspaces of very high-dimensional data, under various conditions of cluster overlapping. It is known from previous research that the performance of clustering algorithms of k-means type generally depend on whether the data set contains well separated clusters, which motivates tests in various scenarios.

We generate high-dimensional data sets with clusters embedded in different subspaces using a data generator, derived from, but not identical to, the generation algorithm used in [15]. The algorithm used is summarized in appendix C. In order to better measure the difficulties of the generated data sets, three parameters are explicitly controlled in the generator. In the description of these parameters, we assume that the (sub)space of a cluster consists of its relevant dimensions.

subspace ratio The *subspace ratio*, ε, of a cluster is the average ratio of the dimension of the subspace to that of the whole space, i.e. it determines the average size of the subspace of each cluster and lies in the range $\varepsilon \in [0, 1]$. The subspace ratio is defined mathematically by Jing et al. in [15] as

[2] http://kdd.ics.uci.edu

$$\varepsilon = \frac{1}{km} \sum_{l=1}^{k} m_l,$$

where m_l is the number of relevant dimensions in the lth cluster, m is the total number of dimensions in the data set and k is the number of clusters.

relevant dimension overlap ratio The *(relevant) dimension overlap ratio*, ρ, of a cluster, is also defined in [15]. It is the ratio of the dimension of the overlapping subspace, which also belongs to another cluster, to the dimension of its own subspace. For example, suppose that the subspace of cluster A is $\{2, 5, 8, 9, 14, 17, 18, 22\}$, whereas that of cluster B in the same data set is $\{1, 3, 5, 6, 10, 11, 14, 15, 18\}$. The overlapping subspace A and B is thus $\{5, 14, 18\}$ and its dimension size is 3. Since the dimension size of the subspace of cluster A is 8, the dimension overlap ratio of cluster A with respect to cluster B is $\rho = 0.375$. According to this definition, each generated cluster is guaranteed to have a dimension overlap ratio ρ with at least one other cluster, for any given ρ. Consequently, the dimension overlap ratio of a generated cluster with respect to any other generated cluster could be either greater or smaller than ρ. Despite this disambiguity, the dimension overlap ratio is still a good measure of the true overlapping if relevant dimensions between clusters in a data set.

data overlap ratio The *data overlap ratio*, α, is a special case of the concept of *overlapping rate* that was proposed in [44] and [45]. In these articles, the overlapping rate between two Gaussian clusters was defined as the ratio of the minimum of the mixture probability density function (pdf) on the ridge curve linking the centers of two clusters to the height of the lower peak of the pdf. Although complex in nature, it has been shown that this measure is a better indicator of the separability between two clusters than the more conventional concept based on the *Bayesian classification error* ([46]). In the one-dimensional case, if the variance of the two clusters is fixed, then the overlap rate is solely determined by the distance between their respective centroids. In the data generation scheme presented in this section, the data overlap ratio is a parameter that controls the distance between two adjacent clusters; the centroid of the cluster currently being generated and the centroid of the cluster generated in the previous step on each of the overlapped relevant dimensions.

Three categories of synthetic data sets are generated. They differ in the number of dimensions, which are 100, 1 000 and 2 000. In these synthetic data sets, each cluster has its own relevant dimensions, which are randomly

determined based on ε and can overlap. The data values are normally distributed on each relevant dimension of each cluster. The corresponding mean is uniformly distributed in the range $[0, 100]$ and the corresponding variance is set to 1. In contrast, the data values for irrelevant dimensions are uniformly distributed in the range $[0, 10]$. The dimension overlap ratio ρ and the data overlap ratio α are chosen from the sets $\{0.25, 0.5, 0.75\}$ and $\{0.5, 1, 2\}$, respectively. The subspace ratio ε is set to 0.375. In general, the larger the value of ρ and the smaller the value of α, the more complicated the synthetic data sets are to cluster, in terms of dimension overlapping and data overlapping in common relevant dimensions. All in all, a total number of 27 data sets are generated. For each category of data, there are 9 data sets, each of which has 500 data objects divided into 10 clusters of size 50. All data sets are generated using MATLAB.

8.2 Parameter settings for algorithms

To guarantee that the two competing algorithms are run under identical conditions, it is crucial to supply each algorithm with the same set of parameters. In most cases we use the same parameter settings that was previously used in the study of Lu et al. ([12]), the developers of PSOVW, that conducted a similar comparison between PSOVW and a few other common soft projected clustering algorithms.

The Euclidean distance is used as the dissimilarity measure between two data objects. The parameter β is set to 2. The swarm size s is set to 10, independently of the dimensionality of the data sets. The inertia weight λ is set to 0.7 and the acceleration factor ϕ is set to 1.45. These parameter settings yield a stable evolution of UPSOVW according to the stability region presented in figure 1. For the PSOVW algorithm, the velocity clamping threshold is set to $v_{\max} = 0.25$, i.e. one-fourth of the allowed range for a variable weight in PSOVW. Although velocity clamping is not employed in UPSOVW, we use this value to constrain the magnitudes of the particle velocities during initialization, so that the velocities effectively start off in the range $[-v_{\max}, v_{\max}]$.

In order to sustain good population diversity, particles in both algorithms are required to have different exploration and exploitation abilities, which is controlled by the P_c probabilities. For this reason it has been proposed that each particle should have a different P_c value. The P_c probabilities are therefore computed according to

$$\begin{aligned} P_c(i) &= \frac{1}{20} + \frac{9}{20} \cdot \frac{e^{f(i)} - 1}{e^{f(s)} - 1}, \\ \text{where } f(i) &= 10 \cdot \frac{i-1}{s-1}, \end{aligned}$$

for all particles $i = 1, 2, \ldots, s$. The formula was empirically determined for CLPSO in [29]. The values of the learning probabilities P_c thus range from 0.05 to 0.5. Another consequence is that a particle with higher particle id is more likely to learn from other personal bests than a particle with lower id. The learning probabilities are presented graphically in figure 2. We use this implementation for computing the P_c values and set the refreshing gap to 5 for both algorithms.

There must also be a fair stopping criterion imposed on the algorithms. At first, it may seem suitable to specify a maximum number of iterations allowed for each run. However, since the particles in PSOVW are only updated if they are within the allowed range, we suspect that PSOVW will perform an iteration much faster than UPSOVW that updates each particle of each iteration. On the other hand, if we specify a maximum number of fitness evaluations, the UPSOVW will finish considerably faster than PSOVW in the same scenario. Taking these aspects into account, it is clear that a fair stopping criterion should be based on the respective running times of the algorithms. We will therefore restrict both algorithms to a running time of 40, 100 and 150 seconds for the data sets with 100, 1 000 and 2 000 dimensions, respectively. In this way the comparisons will be fair as long as each run is carried out on the same machine.

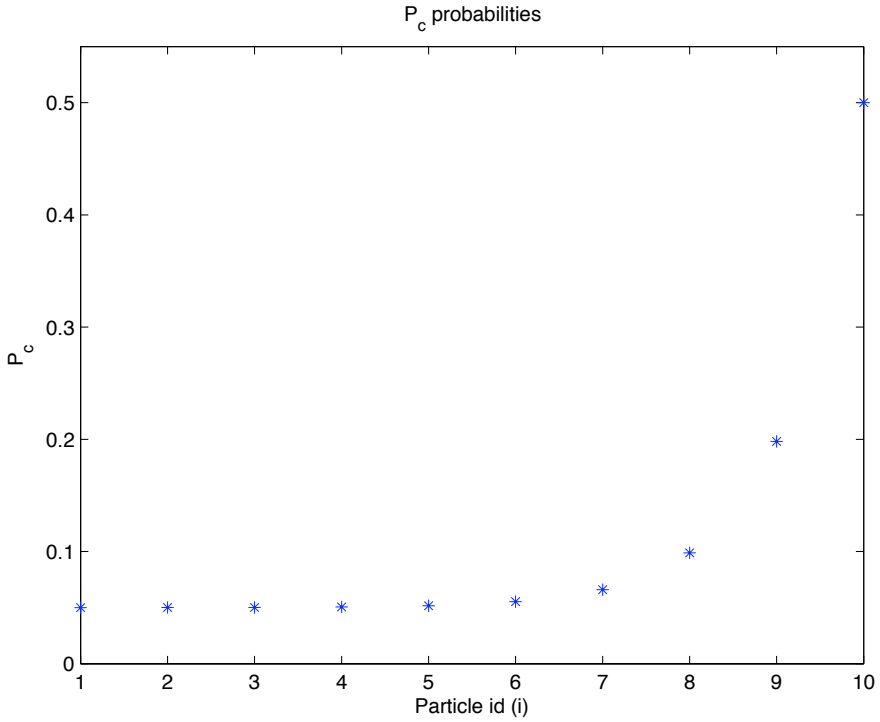

Figure 2: P_c probabilities for a swarm consisting of 10 particles.

8.3 Results for synthetic data

The clustering has been performed on a PowerBook G4 laptop equipped with a 1.5 gigahertz PowerPC G4 processor and 1.25 gigabytes random access memory. The code for the PSOVW algorithm was provided by Yanping Lu, one of the authors of [12], and the UPSOVW algorithm was implemented by the author. Both algorithms were implemented in C++.

Each algorithm is run 10 times on each data set. The rationale for studying a number of trials (instead of just one) lies in the stochastic nature of the algorithms. Because of the random influence, it is plausible that the outcome of each run will differ in some ways. However, if we let the algorithms run for a number of times a pattern should emerge, arguably after no more than 10 runs. Apart from the stochastic effects on the particle updates and initializations, both algorithms rely on randomly initialized cluster centroids. The effect of these initializations are often studied in performance evaluations regarding clustering algorithms of k-means type, since it can serve as a measure of the robustness of the method in scope.

To compare the clustering performance of the algorithms, we first present the average cluster quality. The cluster quality is in general measured by the clustering accuracy, which is the ratio of the data objects that are correctly recovered by an algorithm to the number of objects in the whole data set. However, the problem of computing the clustering accuracy can be a challenging one, since the clustering can yield a very large number of possible outcomes. Correctly identified data objects are also dependent on whether other objects are considered as correct, which dismisses any intuitive idea of how the accuracy computation should be tackled. So, instead of counting the correctly identified objects, one can count the number of erroneous classifications as a measure of an algorithms performance. One such measure is known as the *Classified Error Rate* ([47]), which is defined in the following.

Definition (Classified Error Rate). Let G_l be the original clusters produced by a data generator and let C_l be the clusters produced by a clustering algorithm applied to a data set S, such that

$$\bigcup_{l=1}^{k} G_l = \bigcup_{l=1}^{k} C_l = S,$$

where k is the number of clusters. Define the function δ as

$$\delta(o_1, o_2, C_l) = \begin{cases} 0, & \text{if there exists } j \text{ such that } o_1, o_2 \in G_l \\ 1, & \text{otherwise,} \end{cases}$$

where $o_1, o_2 \in C_l$. Then, the *Classified Error Rate* is defined as

$$\text{CER} = \frac{\sum_{l=1}^{k} \sum_{m,n \in C_l} \delta(m, n, C_l)}{\sum_{l=1}^{k} \binom{N_l}{2}},$$

where N_l is the number of data objects in C_l.

Because clustering algorithms may produce clusters of different sizes, there is no easy way of converting between clustering accuracy and CER. So, we will in the following measure the cluster quality in terms of CER, which is suitable in this context since we are only interested in comparing different CER values. It is also easily computed compared to the clustering accuracy. Apart from this performance measure, the variance of the cluster quality is also important. A low variance implies a small dependence of the initial cluster centroids and the stochastic effects on the final result. However, since PSOVW and UPSOVW are very similar in both of these aspects, we will not include the variances in the study, but solely focus on the CER as a performance measure.

Tables 1, 2 and 3 below show the average classified error rate over 10 runs of the two algorithms for data sets consisting of 100, 1 000 and 2 000 dimensions, respectively, with different values for ρ and α.

Table 1 Average Classified Error Rate of PSOVW and UPSOVW over 10 trials on each data set with 100 dimensions

$\rho \setminus \alpha$	0.5	1	2	Algorithm
0.25	**0.1773**	**0.2580**	0.0594	PSOVW
	0.1938	0.2686	**0.0391**	UPSOVW
0.5	**0.2443**	0.2176	0.1996	PSOVW
	0.4689	**0.1806**	**0.1405**	UPSOVW
0.75	**0.4011**	0.3255	**0.3927**	PSOVW
	0.6114	**0.2631**	0.3928	UPSOVW

Table 2 Average Classified Error Rate of PSOVW and UPSOVW over 10 trials on each data set with 1 000 dimensions

$\rho \setminus \alpha$	0.5	1	2	Algorithm
0.25	**0.0834**	**0.0871**	**0.1236**	PSOVW
	0.1886	0.1819	0.1313	UPSOVW
0.5	**0.1478**	**0.1041**	**0.2265**	PSOVW
	0.2584	0.2641	0.2388	UPSOVW
0.75	**0.1312**	**0.2410**	**0.1046**	PSOVW
	0.2819	0.2864	0.2004	UPSOVW

Table 3 Average Classified Error Rate of PSOVW and UPSOVW over 10 trials on each data set with 2 000 dimensions

$\rho \setminus \alpha$	0.5	1	2	Algorithm
0.25	**0.1228**	**0.1052**	**0.1561**	PSOVW
	0.1524	0.1613	0.2201	UPSOVW
0.5	**0.1848**	**0.1579**	**0.1804**	PSOVW
	0.2919	0.2660	0.2977	UPSOVW
0.75	0.3192	**0.2253**	**0.1206**	PSOVW
	0.3089	0.2931	0.3104	UPSOVW

A number of observations can be made from the results in Table 1, 2 and 3. Firstly, PSOVW seems to perform better than UPSOVW on most of the data sets. In fact, it surpasses UPSOVW on 22 of the 27 data sets. This is reflected by a lower CER value, which is indicated by bold face typesetting. In some cases, the differences in cluster quality is negligible, and in others, it is clearly in favour of PSOVW. Secondly, there is a clear pattern in the performance of the algorithms that can be attributed to the variations in the complexity of the data sets. Clearly, the cluster quality seems to drop with increasing dimension overlap ratio ρ, with a few minor exceptions. On the other hand, it was expected that the results would reveal a strong connection between the cluster quality and the data overlap ratio α, but there seems to be no trend that promotes such a conclusion. This is quite a surprising fact, since larger α corresponds to more separated clusters that should be more easily identifiable by the algorithms.

Moreover, we initially suspected a better performance from UPSOVW, because of its unconstrained search capability. In UPSOVW, each particle is updated at each iteration, contrary to PSOVW. This is reflected by the number of iterations performed by each algorithm until the stopping criterion is met. On average, PSOVW carries out in between two and four times as many iterations as UPSOVW, depending on the test case. Despite the superior performance of PSOVW in terms of clustering accuracy, the minimum fitness value recovered by UPSOVW is in fact smaller than that of PSOVW. Table 4 through 6 present the average minimum for the two algorithms on each data set. Notably, UPSOVW outperforms PSOVW when it comes to optimizing the objective function on all 27 data sets, although PSOVW seems to perform better in terms of clustering accuracy.

Table 4 Average minimum fitness value of PSOVW and UPSOVW over 10 trials on each data set with 100 dimensions

$\rho \setminus \alpha$	0.5	1	2	Algorithm
0.25	3113.1	3095.1	3072.8	PSOVW
	2911.6	**2918.8**	**2886.4**	UPSOVW
0.5	3091.7	3156.3	3117.4	PSOVW
	1961.9	**2918.0**	**2900.1**	UPSOVW
0.75	3023.0	3124.9	2991.6	PSOVW
	1912.0	**2902.5**	**2815.0**	UPSOVW

Table 5 Average minimum fitness value of PSOVW and UPSOVW over 10 trials on each data set with 1 000 dimensions

$\rho \setminus \alpha$	0.5	1	2	Algorithm
0.25	343.1	343.4	339.7	PSOVW
	323.0	**329.4**	**320.1**	UPSOVW
0.5	343.1	341.7	342.2	PSOVW
	322.2	**326.1**	**323.4**	UPSOVW
0.75	341.5	340.6	340.2	PSOVW
	322.5	**315.8**	**319.6**	UPSOVW

Table 6 Average minimum fitness value of PSOVW and UPSOVW over 10 trials on each data set with 2 000 dimensions

$\rho \setminus \alpha$	0.5	1	2	Algorithm
0.25	172.9	173.6	174.0	PSOVW
	166.6	**167.0**	**165.8**	UPSOVW
0.5	172.9	172.4	173.7	PSOVW
	165.5	**166.0**	**164.8**	UPSOVW
0.75	170.5	171.2	171.1	PSOVW
	164.1	**162.2**	**163.7**	UPSOVW

9 Conclusions and future work

As a response to the increasing demand for data analysis tools that can handle high-dimensional data sets efficiently, this thesis has proposed an algorithm for the variable weighting problem in soft projected clustering, called UPSOVW. It is derived from a recent algorithm called PSOVW that has shown some promising results in relation to traditional clustering methods.

The choice of objective function and search strategy is crucial to any soft projected clustering algorithm. PSOVW utilizes particle swarm optimization to find optimal weights in a k-means weighting function. Despite good clustering accuracy, it has been argued in previous work that the PSOVW algorithm is time-consuming in comparison to other projected clustering algorithms. UPSOVW has been developed to speed up the convergence rate of PSOVW by refining its search capability. In order to do so, we have proposed a modified objective function that excludes all bound constraints on the variable weights by normalizing the weight matrix. The objective function effectively coincides with that of PSOVW as it also calculates the sum of the within-cluster distance for each cluster along relevant dimensions in preference to irrelevant ones. The main difference between the two is that UPSOVW permits search in the whole space in quest for a global minimum, in contrast to PSOVW that is restrained by bound constraints. The normalization technique is not restricted uniquely to this type of objective function, but can readily be applied to any optimization problem that meets certain requirements.

We have compared the performance of the two algorithms on a variety of high-dimensional synthetic data sets. The study shows that PSOVW yields better clustering accuracy than UPSOVW, although the differences are small under many circumstances. None of the algorithms perform perfectly in terms of clustering accuracy on the whole, which implies that they fail to recover the relevant dimensions totally. The dimensionality of the data sets and the number of relevant dimensions for each cluster are extremely high compared to the number of data points, resulting in very sparse data sets. This might a reason why the algorithms fail to recover all relevant dimensions.

Although PSOVW performs better than UPSOVW in terms of clustering accuracy, UPSOVW is superior to PSOVW when it comes to minimizing the objective function. This can be explained by the unconstrained search strategy employed in UPSOVW, which is especially advantageous if the global minimum happens to be in close proximity to a vertex resulting from the bound constraints used in PSOVW. In PSOVW, whenever a particle is near optima of this kind, there is always the possibility that the particle will try to fly outside of the allowed search space, resulting in a lost update for that particle. It is therefore more difficult for PSOVW to home in on these

optima.

Because of the improved search strategy, the fact that UPSOVW yields poorer clustering accuracy than PSOVW is an ambiguous result, since smaller minima should be reflected by more accurate clustering. One explanation might be that the scheme for updating cluster centroids is not well-suited for UPSOVW. Both the membership update mechanism and the fitness function use the Euclidean distance, in contrast to the centroid update scheme which computes the mean of all data points in the corresponding cluster. Instead, one possibility could be to compute the mean-square of the data objects to be consistent with the other formulas. Further investigations are needed.

Two other issues must be focused on in future research. Firstly, the major issue in UPSOVW, as well as in PSOVW, is the requirement of knowing the number of clusters beforehand. In reality, the structure of the data at hand is completely unknown in general, which makes it very difficult to provide a suitable number of clusters in real-world data mining applications. Although a number of attempts have been made to determine the number of clusters automatically in recent years (see for instance [48] and [49]), there exists no successful method up to date, in particular for high-dimensional data sets.

Secondly, since the performance of UPSOVW largely depends on an unconstrained searching technique, it is crucial that the internal parameters of the algorithm ensure a stable behaviour. The underlying optimization method, CLPSO without the effect of velocity clamping, is heavily influenced by the acceleration factor and the inertia weight. In this thesis, we have conducted a stability analysis that accounts for the stochastic forces acting on the particle dynamics, but have done so under a few simplifying assumptions. A more accurate analysis is needed to give clear directions for parameter choices ensuring stability. On the other hand, stability is not the sole interest. Rather, a comprehensive analysis should provide guidelines for parameter choices that enhance performance while sustaining stability of the method. In general, to improve the search capability of a global optimization algorithm, such as CLPSO, it is imperative to find a balance between exploration and exploitation. The L^2 *gain* can be interpreted as a measure of the exploration ability of the CLPSO algorithm, while the *decay rate* can be interpreted as the exploitation ability or the convergence speed of the algorithm. A study of these measures with an emphasis on classic PSO is carried out by Wakasa et al. in [40]. The computation of the L^2 gain and the decay rate are casted in the form of LMIs. A simple bisection algorithm can then be applied to each LMI to find the smallest upper bound on the L^2 gain and the largest lower bound on the decay rate. Of course, this idea can easily be altered to hold for the CLPSO algorithm as well.

A Proofs

A.1 Proof of main theorem

Firstly, we show that the bound constraints in (5.2) are satisfied for $W\prime$. Clearly, we have that all entries $w_{l,j}\prime$ are nonnegative. Morever, because of the definition of $w^*_{\max}$, we have $\left|w^*_{l,j}\right| \leq w^*_{\max}$, for all entries, and so

$$w_{l,j}\prime = \frac{\left|w^*_{l,j}\right|}{w^*_{\max}} \leq 1, \quad 1 \leq l \leq k, 1 \leq j \leq m.$$

Hence, the bound constraints are satisfied.

It remains to show that $W\prime$ is a minimizer of F as defined in (5.1). By definition, we have

$$\begin{aligned}
F(W\prime) &= \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{m} u_{l,i} \cdot \left(\frac{w_{l,j}\prime}{\sum_{s=1}^{m} w_{l,s}\prime}\right)^{\beta} \cdot d(x_{i,j}, z_{l,j}) \\
&= \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{m} u_{l,i} \cdot \left(\frac{\left|w^*_{l,j}\right| / w^*_{\max}}{\sum_{s=1}^{m}\left|w^*_{l,s}\right| / w^*_{\max}}\right)^{\beta} \cdot d(x_{i,j}, z_{l,j}) \\
&= \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{m} u_{l,i} \cdot \left(\frac{w^*_{\max}\left|w^*_{l,j}\right|}{w^*_{\max}\sum_{s=1}^{m}\left|w^*_{l,s}\right|}\right)^{\beta} \cdot d(x_{i,j}, z_{l,j}) \\
&= \sum_{l=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{m} u_{l,i} \cdot \left(\frac{\left|w^*_{l,j}\right|}{\sum_{s=1}^{m}\left|w^*_{l,s}\right|}\right)^{\beta} \cdot d(x_{i,j}, z_{l,j}) \\
&= G(W^*) \\
&\leq G(W),
\end{aligned}$$

for all $W \in \mathbf{R}^{k\times m}$, since W^* is a minimizer of G. Finally, this holds for all real-valued matrices and must therefore hold for all matrices with non-negative entries. However, from the definitions of F and G follow that they coincide for these matrices, which shows that

$$G(W) = F(W),$$

for all matrices that satisfy the bound constraints in (5.2). Hence, we have shown that $F(W\prime) \leq F(W)$, for all matrices W that meet the constraints, and the proof is complete.

A.2 Proof of stability criterion

We now prove that condition (7.5) is a sufficient and necessary condition for mean-square stability. Firstly, we prove that condition (7.5) is sufficient. Let $P > 0$ satisfy (7.5). Moreover, introduce the linear Lyapunov function $V(M) = \text{trace}(MP)$, which is positive on the cone of nonnegative matrices, and can hence serve as a Lyapunov candidate function. For $M(t), t \geq 0$, satisfying (7.3), we then have that

$$\begin{aligned} V\left(M(t+1)\right) &= \text{trace}\left(\left[A_0 M(t) A_0^T + \sigma^2 A_p M(t) A_p^T\right] P\right) \\ &= \text{trace}\left(A_0 M(t) A_0^T P\right) + \sigma^2 \cdot \text{trace}\left(A_p M(t) A_p^T P\right) \\ &= \text{trace}\left(M(t) A_0^T P A_0\right) + \sigma^2 \cdot \text{trace}\left(M(t) A_p^T P A_0\right) \\ &= \text{trace}\left(M(t)\left[A_0^T P A_0 + \sigma^2 A_p^T P A_0\right]\right), \end{aligned}$$

where we have used the linear relationship, $\text{trace}\,(\alpha A + \beta B) = \alpha \cdot \text{trace}\,(A) + \beta \cdot \text{trace}\,(B)$, and the identity $\text{trace}\,(AB) = \text{trace}\,(BA)$, that both hold for all square matrices. Further, since P satisfies (7.5), we have that

$$\text{trace}\left(M(t)\left[A_0^T P A_0 + \sigma^2 A_p^T P A_0\right]\right) < \text{trace}\,(M(t)P),$$

for all nonzero $M(t)$ satisfying (7.3). Hence, $V\left(M(t+1)\right) < V\left(M(t)\right)$, for all $t \geq 0$, which shows that $M(t)$ converges to zero as $t \to \infty$ according to a standard argument from Lyapunov theory.

To prove that condition (7.5) is also necessary, we assume that the system is mean-square stable. By the definition of mean-square stability (7.4), this implies that for every positive-semidefinite initial condition $M(0) \geq 0$, the solution to the linear recursion (7.3) goes to zero as $t \to \infty$. In fact, the linearity of system (7.3) implies that, for an arbitrary choice of $M(0)$, the corresponding solution $M(t)$ converges to zero, that is, (7.3) is stable. We now rewrite (7.3) as a linear system $m(t+1) = \mathcal{A}m(t)$, where $m(t)$ is a vector containing the n^2 elements of $M(t)$ and $\mathcal{A}$ is a real matrix expressed via Kronecker products:

$$\mathcal{A} = A_0 \otimes A_0 + \sigma^2 \cdot A_p \otimes A_p.$$

Now, stability of $\mathcal{A}$ is equivalent to that of $\mathcal{A}^T$. By a standard result concerning Kronecker products, namely,

$$\begin{aligned} \mathcal{A}^T &= \left(A_0 \otimes A_0 + \sigma^2 \cdot A_p \otimes A_p\right)^T \\ &= A_0^T \otimes A_0^T + \sigma^2 \cdot A_p^T \otimes A_p^T, \end{aligned}$$

we see that stability of (7.3) is equivalent to that of the matrix difference equation

$$N(t+1) = A_0^T N(t) A_0 + \sigma^2 A_p^T N(t) A_p. \tag{A.1}$$

Further, let $N(0) > 0$ and let $N(t)$ be the corresponding solution to (A.1). Since $N(k)$ satisfies a stable difference equation of first order with constant coefficients, the function

$$P(t) = \sum_{i=1}^{t} N(i)$$

has a finite limit as $t \to \infty$, which we call P. From $N(0) > 0$ follows that $P > 0$. The recursion (A.1) implies

$$P(t+1) = N(0) + A^T P(t) A + \sigma^2 A_p^T P(t) A_p.$$

Taking the limit $t \to \infty$ shows that P satisfies (7.5).

B A short introduction to LMIs

A linear matrix inequality (LMI) has the form

$$F(x) = F_0 + \sum_{i=1}^{m} x_i F_i > 0, \tag{B.1}$$

where $x \in \mathbf{R}^m$ is the variable and the matrices $F_i \in \mathbf{R}^{n \times n}, i = 0, 1, \ldots, m$, are symmetric and given beforehand. The inequality symbol means that $F(x)$ is positive-definite, that is, $v^T F(x) v > 0$, for all nonzero $v \in \mathbf{R}^n$. This LMI is known as *strict*, in contrast to a *nonstrict* LMI, which has the form

$$F(x) \geq 0.$$

The strict and the nonstrict LMI are closely related. In fact, it turns out that a feasible nonstrict LMI can be reduced to an equivalent strict LMI in most cases. The reduction is in principle carried out by eliminating implicit equality constraints and then reducing the resulting LMI by removing any constant nullspace.

The strict LMI in (B.1) implies a convex constraint on x, i.e. the set $\{x \colon F(x) > 0\}$ is convex. Given an LMI $F(x) > 0$, the corresponding LMI problem (LMIP) is to find a feasible vector x_f such that $F(x_f) > 0$ or determine that there is no such vector, in which case we say that the LMI is infeasible.

Although the LMI in (B.1) may appear to be of a specialized form, it can in fact be applied to numerous problems that present convex constraints on x. For instance, linear inequalities as well as matrix norm inequalities can be expressed as LMIs. In particular, many Lyapunov and convex quadratic matrix inequalities arise in the form of LMIs, which is the major reason

for studying LMIs in the context of control theory and stability analysis of dynamical systems.

Beside the flexibility of the standard form (B.1), it permits a straightforward treatment of systems of LMIs. In particular, given a number of LMIs $F_1(x) > 0, \ldots, F_p(x) > 0$ can be cast in the form of the single LMI $\mathbf{diag}\,(F_1(x), \ldots, F_p(x)) > 0$.

Some LMI problems are so frequently encountered in various situations that they are often referred to as standard problems. One is the obvious LMIP $F(x) > 0$ mentioned above, whereas the others are briefly explained in the following:

eigenvalue problems The eigenvalue problem (EVP) is to minimize the maximum eigenvalue of a matrix that depends affinely on a variable, subject to an LMI constraint. The general form of an EVP is

$$\begin{array}{rl} \min & \lambda \\ \text{subject to} & \lambda I - A(x) > 0, \quad B(x) > 0, \end{array}$$

where A and B are symmetric matrices that are affine functions of the variable x, yielding a convex optimization problem.

generalized eigenvalue problems The generalized eigenvalue problem (GEVP) is closely related to the EVP. A GEVP is to minimize the maximum generalized eigenvalue of a pair of matrices that are affine functions of a variable, subject to an LMI constraint. More formally, this can be formulated as

$$\begin{array}{rl} \min & \lambda \\ \text{subject to} & \lambda B(x) - A(x) > 0, \quad B(x) > 0, \quad C(x) > 0, \end{array}$$

where A, B and C are symmetric matrices that depend affinely on x.

matrix determinant problems Although the previous problems are the most common, sometimes one encounters the following problem, which we abbreviate MDP:

$$\begin{array}{rl} \min & \log\det A(x)^{-1} \\ \text{subject to} & A(x) > 0, \quad B(x) > 0. \end{array}$$

As usual, A and B are symmetric matrices that are affine functions of x. This is a convex optimization problem, since when $A > 0$, $\log\det A^{-1}$ is a convex function of A.

The standard problems are both tractable from a mathematical and a practical viewpoint, in that they can be solved in polynomial-time. In this context, finding a solution means to determine whether or not the problem is feasible, and if it is, compute a feasible point according to a desired accuracy. There are several algorithms that can solve these problems efficiently, one being the *ellipsoid algorithm* ([50]), which is suitable because of its simplicity. However, in practice, interior-point algorithms, such as the *method of centers* ([51]), are far more efficient.

C Generating synthetic data sets

The algorithm for generating a synthetic data set is derived from but not identical to the one presented by Lu et al. in [12], which was used to compare the performance of PSOVW to similar algorithms arising in soft projected clustering. This data generation algorithm was in turn inspired by the one presented by Jing et al. in [15], that was used to test the performance of the EWKM algorithm.

- *Set parameters*

  Specify the number of clusters k, the number of dimensions m and the number of objects n. Set the parameters ε (the subspace ratio), ρ (the dimension overlap ratio), α (the data overlap ratio) and σ (the variance for relevant dimensions).

- *Determine relevant dimensions for each cluster*

  For each cluster q,

  Assign a random integer m_q in the range $[2, m]$, such that $\sum m_q = \varepsilon \cdot k \cdot m$.

  End

  For cluster 1,

  Randomly choose m_1 relevant dimensions for C_1.

  For each cluster $q \geq 2$,

  Randomly choose $\rho \cdot m_q$ relevant dimensions from the relevant dimensions of C_{q-1}.

  Randomly choose $(1 - \rho) \cdot m_q$ relevant dimensions from the other dimensions.

  End

- *Generate the mean μ for each relevant dimension of each cluster*

  For cluster 1,

  Randomly set $\mu_{1,j}$ with a uniform distribution in the range $[0, 100]$.

  For each cluster $q \geq 2$,

  If the relevant dimension j is not a common relevant dimension of clusters C_q and C_{q-1},

  Randomly set $\mu_{q,j}$ with a uniform distribution in the range $[0, 100]$.

  Otherwise,

  If $\mu_{q-1,j} + \alpha \cdot \sigma > 100$,

  $\mu_{q,j} = \mu_{q-1,j} - \alpha \cdot \sigma$.

  Otherwise,

  $\mu_{q,j} = \mu_{q-1,j} + \alpha \cdot \sigma$.

  End

  End

  End

- *Generate data points for each cluster*

  For each cluster q,

  For each dimension j,

  If j is a relevant dimension of C_q,

  Produce the data points with a normal distribution $N(\mu_{q,j}, \sigma)$.

  Otherwise,

  Produce the data points with a uniform distribution in the range $[0, 10]$.

  End

  End

  End

References

[1] Chan, P., Fan, W., Prodromidis, A., & Stolfo, S. (1999) Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems, Vol. 14, No. 6*, pp. 67 - 74.

[2] Bass, T. (2000) Intrusion detection systems and multisensor data fusion. *Communications of the ACM, Vol. 43, No. 4*, pp. 99 - 105.

[3] Smith, K. A., Willis, R. J., & Brooks, M. (2000) An analysis of customer retention and insurance claim patterns using data mining: A case study. *The Journal of the Operational Research Society, Vol. 51, No. 5*, pp. 532 - 541.

[4] van der Putten, P., & van Someren, M. (eds) (2000) CoIL Challenge 2000: the insurance company case. Published by *Sentient Machine Research*, Amsterdam. Technical Report.

[5] Kriegel, H.-P., Kroger, P., & Zimek, A. (2009) Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery Data, Vol. 3, No. 1*, pp. 1 - 58.

[6] Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (2005) Automatic subspace clustering of high-dimensional data. *Data Mining and Knowledge Discovery, Vol. 11, No. 1*, pp. 5 - 33.

[7] Goil, G. S., Nagesh, H., & Choudhary, A. (1999) Mafia: Efficient and scalable subspace clustering for very large data sets. *Technical Report CPDC-TR-9906-010*, Northwestern University.

[8] Moise, G., & Sander, J. (2008) Finding non-redundant, statistically significant regions in high-dimensional data: A novel approach to projected and subspace clustering. *Proceeding of the 14th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 533 - 541.

[9] Procopiuc, C. M., Jones, M., Agarwal, P. K., & Murali, T. M. (2002) A Monte Carlo algorithm for fast projective clustering. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 418 - 427.

[10] Woo, K.-G., Lee, J.-H., Kim, M.-H., & Lee, Y.-J. (2004) FINDIT: A fast and intelligent subspace clustering algorithm using dimension voting. *Information and Software Technology, Vol. 46, No. 4*, pp. 255 - 271.

[11] Achtert, E., Böhm, C., Kriegel, H.-P., Kröger, P., Müller-Gorman, I., & Zimek, A. (2007) Detection and visualization of subspace cluster hierarchies. *Lecture Notes in Computer Science, Vol. 4443*, pp. 152 - 163. Berlin: Springer.

[12] Lu, Y., Wang, S., Li, S., & Zhou, C. (2009) Particle swarm optimizer for variable weighting in clustering high-dimensional data. *Swarm Intelligence Symposium, 2009*, pp. 37 - 44.

[13] Domeniconi, C., Gunopulos, D., Ma, S., Yan, B., Al-Razgan, M., & Papadopoulos, D. (2007) Locally adaptive metrics for clustering high dimensional data. *Data Mining and Knowledge Discovery Journal, Vol. 14, No. 1*, pp. 63 - 97.

[14] Huang, J. Z., Ng, M. K., Rong, H., & Li, Z. (2005) Automated variable weighting in k-means type clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 5*, pp. 657 - 668.

[15] Jing, L., Ng, M. K., & Huang, J. Z. (2007) An entropy weighting k-means algorithm for subspace clustering of high-dimensional sparse data. *IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 8*, pp. 1026 - 1041.

[16] MacQueen, J. B. (1967) Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1*, pp. 281 - 297. University of California Press.

[17] Eberhart, R. C., & Kennedy, J. (1995) A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39 - 43.

[18] Clerc, M. (2006) Stagnation analysis in particle swarm optimisation or what happens when nothing happens. *Technical Report CSM-460*, Department of Computer Science, University of Essex.

[19] Parsopoulos, K. E., & Vrahatis, M. N. (2004) On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation, Vol. 8, No. 3*, pp. 211 - 224.

[20] Blackwell, T., & Bentley, P. J. (2002) Don't push me! Collision-avoiding swarms. *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1691 - 1696.

[21] Eberhart, R. C., Simpson, P. K., & Dobbins, R. W. (1996) *Computational Intelligence PC Tools, first ed.* Academic Press Professional

[22] Shi, Y., & Eberhart, R. C. (1998) A modified particle swarm optimizer. *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 69 - 73.

[23] Eberhart, R. C., & Shi, Y. (2000) Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 84 - 88.

[24] Eberhart, R. C., & Shi, Y. (2001) Tracking and optimizing dynamic systems with particle swarms. *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 94 - 100.

[25] Zheng, Y.-L., Ma, L.-H., Zhang, L.-Y., & Qian, J.-X. (2003) On the convergence analysis and parameter selection in particle swarm optimization. *Proceedings of the IEEE International Conference on Machine Learning and Cybernetics*, pp. 1802 - 1807.

[26] Koziel, S., & Michalewicz, Z. (1999) Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation, Vol. 7, No. 1*, pp. 19 - 44.

[27] He, Q., & Wang, L. (2007) An effective co-evolutionary particle swarm optimization for constrained engineering design problems. *Engineering Applications of Artificial Intelligence, Vol. 20, No. 1*, pp. 89 - 99.

[28] Ray, T., & Liew, K.M. (2001) A swarm with an effective information sharing mechanism for unconstrained and constrained single objective optimisation problems. *Proceedings of the 2001 Congress on Evolutionary Computation, Vol. 1*, pp. 75 - 80.

[29] Liang, J. J., Qin, A. K., Suganthan, P. N., & Baskar, S. (2006) Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation, Vol. 10, No. 3*, pp. 281 - 295.

[30] Parsopoulos, K. E., & Vrahatis, M. N. (2004) UPSO - A unified particle swarm optimization scheme. *Lecture Series on Computational Sciences*, pp. 868 - 873.

[31] Ozcan, E., & Mohan, C. K. (1998) Analysis of a simple particle swarm optimization system. *Intelligent Engineering Systems Through Artificial Neural Networks, Vol. 8*, pp. 253 - 258.

[32] Ozcan, E., & Mohan, C. K. (1999) Particle swarm optimization: surfing the waves. *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1939 - 1944.

[33] Clerc, M., & Kennedy, J. (2002) The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation, Vol. 6, No. 1*, pp. 58 - 73.

[34] Campana, E. F., Fasano, G., Peri, D., & Pinto, A. (2006) Particle swarm optimization: Efficient globally convergent modifications. *III European Conference on Computational Mechanics*, p. 12.

[35] Rudolph, G. (1996) Convergence of Evolutionary Algorithms in General Search Spaces. *Proceedings of the Third IEEE Conference on Evolutionary Computation*, pp. 50 - 54.

[36] Poli, R., Langdon, W. B., Clerc, M., & Stephens, C. R. (2007) Continuous optimization theory made easy? Finite-element models of evolutionary strategies, genetic algorithms and particle swarm optimizers. *Lecture Notes in Computer Science. Vol. 4436*, pp. 165 - 193. Berlin: Springer.

[37] Poli, R. (2009) Mean and variance of the sampling distribution of particle swarm optimizers during stagnation. *IEEE Transactions on Evolutionary Computation, Vol. 13, No. 4*, pp. 712 - 721.

[38] Jiang, M., Luoa, Y. P., & Yanga, S. Y. (2007) Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. *Information Processing Letters, Vol. 102, No. 1*, pp. 8 - 16.

[39] Kadirkamanathan, V., Selvarajah, K., & Fleming, P. J. (2006) Stability analysis of the particle dynamics in particle swarm optimizer. *IEEE Transactions on Evolutionary Computation, Vol. 10, No. 3*, pp. 245 - 255.

[40] Wakasa, Y., Tanaka, K., & Akashi, T. (2009) Stability and l^2 gain analysis for the particle swarm optimization algorithm. *American Control Conference*, pp. 1748 - 1753.

[41] Kushner, H. J. (1967) *Stochastic stability and control*. Academic Press.

[42] Willems, J. L. (1973) The circle criterion and quadratic Lyapunov functions for stability analysis. *IEEE Transactions on Automatic Control, Vol. 18, No. 2*, p. 184.

[43] Gahinet, P., Nemirovski, A., Laub, A. J., & Chilali, M. (1994) The LMI control toolbox. *Proceedings of the 33rd IEEE Conference on Decision and Control, Vol. 3*, pp. 2038 - 2041.

[44] Aitnouri, E., Wang, S., & Ziou, D. (2000) On comparison of clustering techniques for histogram pdf estimation. *Pattern Recognition and Image Analysis, Vol. 10, No. 2*, pp. 206 - 217.

[45] Bouguessa, M., Wang, S., & Sun, H. (2006) An objective function approach to cluster validation. *Pattern Recognition Letters, Vol. 27, No. 13*, pp. 1419 - 1430.

[46] Kantarcioğlu, M., Jin, J., & Clifton, C. (2004) When do data mining results violate privacy? *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 599 - 604.

[47] Wang, T., Liu, D.-X., Lin, X.-Z., Sun, W., & Ahmad, G. (2006) Clustering large scale of XML documents. *Lecture Notes in Computer Science, Vol. 3947*, pp. 447 - 455. Berlin: Springer.

[48] Handl, J., & Knowles, J. (2004) Multiobjective clustering with automatic determination of the number of clusters. *Technical Report*, UMIST, Department of Chemistry.

[49] Tibshirani, R., Walther, G., & Hastie, T. (2001) Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology), Vol. 63, No. 2*, pp. 411 - 423.

[50] Shor, N. Z., Kiwiel, K. C., & Ruszcayǹski, A. (1985) *Minimization methods for non-differentiable functions.* New York: Springer.

[51] Liêũ, B.-T., & Huard, P. (1966) La mèthode des centres dans un espace topologique. *Numerische Mathematik, Vol. 8, No. 1*, pp. 56 - 67.