TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Jenni Raitoharju

Multidimensional Particle Swarm Optimization for Machine Learning

Julkaisu 1455 • Publication 1455

Tampere 2017

Tampereen teknillinen yliopisto. Julkaisu 1455
Tampere University of Technology. Publication 1455

Jenni Raitoharju

Multidimensional Particle Swarm Optimization for Machine Learning

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 24th of February 2017, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2017

ISBN 978-952-15-3903-9 (printed)
ISBN 978-952-15-3919-0 (PDF)
ISSN 1459-2045

Abstract

Particle Swarm Optimization (PSO) is a stochastic nature-inspired optimization method. It has been successfully used in several application domains since it was introduced in 1995. It has been especially successful when applied to complicated multimodal problems, where simpler optimization methods, e.g., gradient descent, are not able to find satisfactory results. Multidimensional Particle Swarm Optimization (MD-PSO) and Fractional Global Best Formation (FGBF) are extensions of the basic PSO. MD-PSO allows searching for an optimum also when the solution dimensionality is unknown. With a dedicated dimensional PSO process, MD-PSO can search for optimal solution dimensionality. An interleaved positional PSO process simultaneously searches for the optimal solution in that dimensionality. Both the basic PSO and its multidimensional extension MD-PSO are susceptible to premature convergence. FGBF is a plug-in to (MD-)PSO that can help avoid premature convergence and find desired solutions faster. This thesis focuses on applications of MD-PSO and FGBF in different machine learning tasks.

Multiswarm versions of MD-PSO and FGBF are introduced to perform dynamic optimization tasks. In dynamic optimization, the search space slowly changes. The locations of optima move and a former local optimum may transform into a global optimum and vice versa. We exploit multiple swarms to track different optima.

In order to apply MD-PSO for clustering tasks, two key questions need to be answered: 1) How to encode the particles to represent different data partitions? 2) How to evaluate the fitness of the particles to evaluate the quality of the solutions proposed by the particle positions? The second question is considered especially carefully in this thesis. An extensive comparison of Clustering Validity Indices (CVIs) commonly used as fitness functions in Particle Swarm Clustering (PSC) is conducted. Furthermore, a novel approach to carry out fitness evaluation, namely Fitness Evaluation with Computational Centroids (FECC) is introduced. FECC gives the same fitness to any particle positions that lead to the same data partition. Therefore, it may save some computational efforts and, above all, it can significantly improve the results obtained by using any of the best performing CVIs as the PSC fitness function.

MD-PSO can also be used to evolve different neural networks. The results of training Multilayer Perceptrons (MLPs) using the common Backpropagation (BP) algorithm and a global technique based on PSO are compared. The pros and cons of BP and (MD-)PSO in MLP training are discussed. For training Radial Basis Function Neural Networks (RBFNNs), a novel technique based on class-specific clustering of the training samples is introduced. The proposed approach is compared to the common input and input-output clustering approaches and the benefits of using the class-specific approach are experimentally demonstrated. With the class-specific approach, the training complexity is reduced, while the classification performance of the trained RBFNNs may be improved.

Collective Network of Binary Classifiers (CNBC) is an evolutionary semantic classifier consisting of several Networks of Binary Classifiers (NBCs) trained to recognize a certain semantic class. NBCs in turn consist of several Binary Classifiers (BCs), which are trained for a certain feature type. Thanks to its topology and the use of MD-PSO as its evolution technique, incremental training can be easily applied to add new training items, classes, and/or features.

In feature synthesis, the objective is to exploit ground truth information to transform the original low-level features into more discriminative ones. To learn an efficient synthesis for a dataset, only a fraction of the data needs to be labeled. The learned synthesis can then be applied on unlabeled data to improve classification or retrieval results. In this thesis, two different feature synthesis techniques are introduced. In the first one, MD-PSO is directly used to find proper arithmetic operations to be applied on the elements of the original low-level feature vectors. In the second approach, feature synthesis is carried out using one-against-all perceptrons. In the latter technique, the best results were obtained when MD-PSO was used to train the perceptrons.

In all the mentioned applications excluding MLP training, MD-PSO is used together with FGBF. Overall, MD-PSO and FGBF are indeed versatile tools in machine learning. However, computational limitations constrain their use in currently emerging machine learning systems operating on Big Data. Therefore, in the future, it is necessary to *divide* complex tasks into smaller subproblems and to *conquer* the large problems via solving the subproblems where the use of MD-PSO and FGBF becomes feasible. Several applications discussed in this thesis already exploit the *divide-and-conquer* operation model.

Preface

This study was carried out at MUVIS group at the Department of Signal Processing at Tampere University of Technology (TUT) during 2010-2016. My research was funded by TUT rector's graduate school 2010-2014. I received additional support for conference costs from Tampere Graduate School in Information Science and Engineering (TISE) and additional financial support from Nokia Foundation and Tekniikan edistämissäätiö (TES). I was using computational resources from TUT's Merope computing cluster and TUT grid (Techila).

I would like to express my sincere gratitude to my supervisor Prof. Moncef Gabbouj for his guidance and support. In his group, I have had the privilege to select my research topics according to my interests. Whenever I have had doubts about my own research, he has been able to return my confidence in it.

I am also thankful to Prof. Serkan Kiranyaz. He has an endless storage of innovative research ideas and it is always motivating to brainstorm these ideas with him. He has significantly contributed to every paper in this thesis. Without him, this thesis would not exist or would be at least very different, as the core content of this thesis, Multidimensional Particle Swarm Optimization, is one of his innovations.

I am grateful to my pre-examiners Prof. Salim Bouzerdoum and Prof. Nikos Nikolaidis for reviewing the thesis and giving me valuable suggestions on how to improve it. I also address special thanks to Prof. Matti Pietikäinen. I'm privileged to have him as my opponent.

I have had the pleasure to work in MUVIS group. The atmosphere in the group has been friendly and supportive. I want to thank the group as a whole and separately Dr. Iftikhar Ahmad, who has been an excellent IT support person for me and helped me to exploit computational resources, and my new office mate Dr. Alexandros Iosifidis, who has shown me an excellent example on how to be a successful postdoctoral researcher. All the way from 2007, when I started as a part-time research assistant, to the day of sending my PhD thesis to pre-examination in October 2016, my working place was in room TC413. In that room, there was always someone able to help me in any practical problems and someone willing to ponder solutions to any emerging problems, whether research-related or not. Special thanks go to my most long-term office mates, Dr. Stefan Uhlmann, Murat Birinci, Guanqun Cao, and Honglei Zhang, who have accompanied me on my way toward the goal and helped me in several ways during the years.

I thank my parents, Pirjo and Erkki, for their support and trust in my skills. During my high school years, my dad used to help me in solving the most difficult tasks in mathematics and physics. I enjoyed our joint problem solving efforts. They changed my opinion on these subjects and thrust me toward studies in a technical university.

During the last few years, my parents have always welcomed my family to their home and volunteered to wake up early with my children to provide me with the luxury of sleeping long.

My beloved husband Matti deserves enormous thanks. First of all, over the years he has concretely helped me with my research, especially in turning the huge amounts of experimental results into meaningful tables and figures. He carefully reviewed this thesis and suggested several improvements from a different viewpoint. More importantly, he has been a wonderful dad to our children. He has equally shared all the parental leaves with me. When needed, I have been able to concentrate on my research and work long hours, as I have always known that my children are in the best possible care in his hands. He has certainly improved the quality of this thesis simply by letting me sleep longer and taking more than his share of early mornings and nightly wake-ups when I was intensively writing the introductory part.

My final thanks go to my lovely children, Annikki and Ilmari. They were both born during the thesis work and they incessantly remind me that there are more important things in my life than the thesis. Sometimes they make me think that my work days are convivial and relaxing, but I always love going back home to see them again.

Tampere, December 2016
Jenni Raitoharju

Contents

Acronyms

ACO	Ant Colony Optimization
ALC-PSO	Particle Swarm Optimization with an Aging Leader and Challengers
ANMRR	Average Normalized Modified Retrieval Rank
ANN	Artificial Neural Network
AP	Average Precision
BC	Binary Classifier
BP	Backpropagation
CBIR	Content-based Image Retrieval
CE	Classification Error
CFS-EM	Co-evolutionary Feature Synthesized Expectation-Maximization
CGP	Co-evolutionary Genetic Programming
CNBC	Collective Network of Binary Classifiers
CNN	Convolutional Neural Network
CVI	Clustering Validity Index
DEPSO	Differential Evolution Particle Swarm Optimization
EA	Evolutionary Algorithm
ECG	Electrocardiogram
ECOC	Error Correcting Output Code
EFS	Evolutionary Feature Synthesis
EM	Expectation-Maximization
EP	Evolutionary Programming
ES	Evolutionary Strategies
FECC	Fitness Evaluation with Computational Centroids
FGBF	Fractional Global Best Formation

GA Genetic Algorithm

GCPSO Guaranteed Convergence Particle Swarm Optimization

GP Genetic Programming

LFPSO Levy Flight Particle Swarm Optimization

MDA Multiple Discriminant Analysis

MD-MPB Multidimensional Moving Peaks Benchmark

MD-PSO Multidimensional Particle Swarm Optimization

MEPSO Multi-Elitist Particle Swarm Optimization

MLP Multilayer Perceptron

MPB Moving Peaks Benchmark

MSE Mean Squared Error

MST Minimum Spanning Tree

NBC Network of Binary Classifiers

NCC Nearest Centroid Classifier

OED Orthogonal Experimental Design

OLPSO Orthogonal Learning Particle Swarm Optimization

PFS Perceptron Feature Synthesis

PSC Particle Swarm Clustering

PSO Particle Swarm Optimization

PSO-MAM Particle Swarm Optimization with Multiple Adaptive Methods

RBF Radial Basis Function

RBFNN Radial Basis Function Neural Network

RF Random Forest

RNN Recurrent Neural Network

SA Simulated Annealing

SAD-PSO Stochastic Approximation -driven Particle Swarm Optimization

SAR Synthetic Aperture Radar

SLP Single-layer Perceptron

SRPSO Self Regulating Particle Swarm Optimization

SVM Support Vector Machine

Nomenclature

Latin alphabet

$a(\cdot)$	Activation function of a neuron
$A(\cdot)$	Peak function in MPB
A_i	i^{th} A-type element in MD-PSO EFS encoding
$b_{\min}$	Minimum number of bits required for a representation
$\mathbf{b}_A$	Artificial global best solution
$\mathbf{b}_A^d$	Artificial global best solution in dimensionality d
$\mathbf{b}_{A,j}^d$	j^{th} element of artificial global best solution in dimensionality d
$\mathbf{b}_p$	Personal best solution of particle p
$\mathbf{b}_p^d$	Personal best solution of particle p in dimensionality d
$\mathbf{b}_S$	Global best solution
$\mathbf{b}_S^d$	Global best solution in dimensionality d
B	Candidate group in MEPSO
$B(\cdot)$	Basis landscape function in MPB
B_j	j^{th} B-type element in MD-PSO EFS encoding
c	Class
c_1, c_2	Acceleration constants in PSO
$\mathbf{c}_q$	Center of peak q in MPB
$\mathbf{c}_q^d$	Center of peak q in dimensionality d in MD-MPB
C	Number of classes
C_j	j^{th} cluster
d	Dimensionality
d^I	Data dimensionality in input clustering
d^{IO}	Data dimensionality in input-output clustering
$d_{\max}$	Maximum dimensionality
$d_{\min}$	Minimum dimensionality
d_{opt}	Optimal dimension in MPB
d_p	Dimensionality of particle p
db_p	Personal best dimensionality of particle p
db_S	Global best dimensionality
dv_p	Dimensional velocity of particle p
e_{kp}	Error at neuron k for pattern p
E_p	Total error for pattern p
f	Feature
$f(\cdot)$	Fitness function for (MD)PSO
$f_{[i]}$	i^{th} element of input feature vector in EFS
f_j	j^{th} feature or feature element
$f_j(\cdot)$	Fractional fitness function for FGBF

$F(\cdot)$	Fitness surface function in MPB
F	Number of features
$g(\cdot)$	Gaussian function
h_q	Height of peak q in MPB
h_k	Parameter of neuron k
i, j, k	Indices
I	Number of items
I^{CS}	Number of items in class-specific clustering
I^I	Number of items in input clustering
I_j	Number of items in cluster C_j
K	Number of clusters
K^{CS}	Number of clusters in class-specific clustering
K^I	Number of clusters in input clustering
$K_{\max}$	Maximum number of clusters
l	Layer index
L	Number of layers
$L_{\max}$	Maximum number of layers when evolving MLPs with MD-PSO
$L_{\min}$	Minimum number of layers when evolving MLPs with MD-PSO
$\mathbf{m}$	Center location of RBF neuron
$\mathbf{m}_k$	Center location of k^{th} RBF neuron
$\mathbf{m}_{p,j}$	j^{th} cluster centroid defined by position of particle p
n	Neighborhood size
N	Number of neurons
N^i	Number of input neurons
N^o	Number of output neurons
$N^l_{\max}$	Maximum number of neurons in layer l when evolving MLPs with MD-PSO
$N^l_{\min}$	Minimum number of neurons in layer l when evolving MLPs with MD-PSO
p	Particle or pattern index
$p[j]$	Index of particle having the best j^{th} element
P	Number of peaks
q	Peak index in MPB
r, r_1, r_2	Random values
r_{bas}	Peak basin radius
r_{rep}	Repulsion radius
$\mathbf{r}, \mathbf{r}_1, \mathbf{r}_2$	Random vectors
R	Number of runs
$R_{\max}$	Maximum range array for evolving MLPs with MD-PSO
$R_{\min}$	Minimum range array for evolving MLPs with MD-PSO
ssw	Within-cluster sum-of-squares
$\mathbf{s}_q$	Shift vector of peak q in MPB
S	Number of particles
t	Iteration
$t_{k,j}$	j^{th} target output element for class c_k
t_{kp}	Target output of neuron k for pattern p
$\mathbf{t}_p$	Target vector for pattern p
T	T-value in Wilcoxon Signed-Ranks test
$T_{p,j}$	Activation threshold for j^{th} centroid defined by particle p in MEPSO
U	Number of operators
$U(0, 1)$	Uniform distribution between 0 and 1

$\mathbf{v}_{\max}$	Maximum velocity
$\mathbf{v}_p$	Velocity of particle p
$\mathbf{v}_p^d$	Velocity of particle p in dimensionality d
w	Weight in EFS or inertia weight in PSO
w_q	Width of peak q in MPB
W	Number of operators used to a synthesize a single feature
$\mathbf{x}$	Position
$\mathbf{x}^d$	Position in dimensionality d
$\mathbf{x}_p$	Position of particle p
$\mathbf{x}_p^d$	Position of particle p in dimensionality d
$\mathbf{x}_{p,j}^d$	j^{th} element of position of particle p in dimensionality d
X	Search space extent
y_j	j^{th} synthesized feature element
y_k^l	Output of neuron k in layer l
y_{kp}^l	Output of neuron k in layer l for pattern p
y_{kp}^o	Output of neuron k in output layer for pattern p
$\mathbf{y}$	Synthesized feature vector
$\mathbf{y}(j)$	j^{th} synthesized feature vector
z_j	j^{th} input element
$\mathbf{z}$	Data item or input vector
$\mathbf{z}_{[i]}$	i^{th} closest data item
$\mathbf{z}_p$	p^{th} input vector
$\mathbf{z}_p^{IO}$	p^{th} input vector for input-output clustering

Greek alphabet

α	Power parameter used in EFS fitness evaluation
β	Weight for output in input-output clustering
β_p	Growth rate of particle p in MEPSO
γ	Weight used in defining width of RBF neuron
Δh	Height change severity in MPB
Δt	Change interval in MPB
Δw	Width change severity in MPB
η	Learning rate in BP
θ_k^l	Bias of neuron k in layer l
$\{\theta_k^l\}$	Set of biases in layer l
Θ_i	i^{th} operator
λ	Correlation factor in MPB
Λ	Shift length in MPB
σ	Width of RBF neuron
σ_k	Width of k^{th} RBF neuron
ω_{jk}^l	Connection weight between neuron k in layer l and its j^{th} input
$\{\omega_{jk}^l\}$	Set of connection weights in layer l

List of Publications

I Turker Ince, Serkan Kiranyaz, Jenni Pulkkinen*, Moncef Gabbouj, "Evaluation of Global and Local Training Techniques over Feed-forward Neural Network Architecture Spaces for Computer-aided Medical Diagnosis," *Expert Systems with Applications*, vol. 37, no. 12, pp. 8450 – 8461, Dec. 2010.

II Serkan Kiranyaz, Stefan Uhlmann, Jenni Pulkkinen*, Moncef Gabbouj, Turker Ince, "Collective Network of Evolutionary Binary Classifiers for Content-based Image Retrieval," *IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS 2011)*, pp. 147 – 154, Apr. 2011.

III Serkan Kiranyaz, Jenni Pulkkinen*, Turker Ince, Moncef Gabbouj, "Multi-dimensional Evolutionary Feature Synthesis for Content-based Image Retrieval," *IEEE International Conference on Image Processing (ICIP 2011)*, pp. 3645 – 3648, Sep. 2011.

IV Serkan Kiranyaz, Jenni Pulkkinen*, Moncef Gabbouj, "Multi-dimensional Particle Swarm Optimization in Dynamic Environments," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2212 – 2223, Mar. 2011.

V Jenni Raitoharju, Serkan Kiranyaz, Moncef Gabbouj, "Evolutionary Feature Synthesis by Multi-dimensional Particle Swarm Optimization," *European Workshop on Visual Information Processing (EUVIP 2014)*, pp. 1 – 6, Dec. 2014.

VI Jenni Raitoharju, Serkan Kiranyaz, Moncef Gabbouj, "Feature Synthesis for Image Classification and Retrieval via One-against-all Perceptrons," *Neural Computing and Applications*, pp. 1 – 15, 2016.

VII Jenni Raitoharju, Serkan Kiranyaz, Moncef Gabbouj, "Training Radial Basis Function Neural Networks for Classification via Class-Specific Clustering," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 12, pp. 2458 – 2471, Dec. 2016.

VIII Jenni Raitoharju, Kaveh Samiee, Serkan Kiranyaz, Moncef Gabbouj, "Particle Swarm Clustering Fitness Evaluation with Computational Centroids," Submitted to *Swarm and Evolutionary Computation*, Aug. 2016.

*The former last name of Jenni Raitoharju was Pulkkinen.

1 Introduction

Over the years, numerous definitions of machine learning have been given. Some of the most quoted include

> Field of study that gives computers the ability to learn without being explicitly programmed (Arthur Samuel, 1959)

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. (Tom Mitchell, 1997)

Tom Mitchell defined the objectives of machine learning research as

> The field of Machine Learning seeks to answer the question: How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?
> (Tom Mitchell, 2006)

Typically, machine learning tasks can be divided into supervised and unsupervised learning. In supervised learning, the desired outputs are given to the algorithm and the objective is to find a general rule that maps the inputs to the desired outputs. In unsupervised learning, the desired output is not available, but the objective is to find some internal data structures. Classification and clustering are perhaps the most obvious examples of supervised and unsupervised learning, respectively. Semi-supervised learning typically exploits a small amount of labeled data and complements the learning using a large amount of unlabeled data.

To succeed in machine learning, it is usually necessary to solve complex optimization tasks in a high-dimensional and multimodal search space. Due to local optima and a huge search space, deterministic gradient-descent-based optimization methods often cannot find proper solutions. Thus, stochastic alternatives are needed. Several stochastic optimization algorithms are inspired by natural phenomena such as genetics (e.g., Genetic Algorithm (GA) [41]) or behavior of animal populations (e.g., Ant Colony Optimization (ACO) [31], Particle Swarm Optimization (PSO) [59]). The common factor of these algorithms is that they strive for better results using an intelligent strategy while preserving some randomness in the process. This randomness can help them escape local optima and find the global optimum faster than their deterministic counterparts. The PSO algorithm mimics the interactions of an animal swarm, e.g., a bird flock or a fish school, in its search for food. The individuals are attracted toward the best feeding areas found by the

other swarm members, but simultaneously they perform their own search in a stochastic manner and may discover even better areas.

This thesis seeks to answer the questions posed by Tom Mitchell when Multidimensional Particle Swarm Optimization (MD-PSO) and Fractional Global Best Formation (FGBF) [67], which are extensions of the basic PSO, are used as tools to learn. That is, the thesis builds machine learning techniques using MD-PSO and FGBF and tries to understand the laws governing the learning processes to further improve the processes. The considered machine learning tasks include dynamic optimization, clustering, classification, image retrieval, and feature synthesis.

1.1 Thesis Objectives

The objective of this thesis was to explore the ability of MD-PSO and FGBF to solve different machine learning tasks and to modify the algorithms or applications in order to obtain a maximal benefit. More specific objectives for different applications were:

- to develop an algorithm suitable for dynamic optimization by combining MD-PSO and FGBF with multiswarms
- to improve Particle Swarm Clustering (PSC) by investigating different options for fitness evaluation
- to analyze the suitability of MD-PSO and FGBF for training neural networks and to improve training of Radial Basis Function Neural Network (RBFNN) by exploiting the *divide-and-conquer* paradigm
- to use MD-PSO and FGBF to learn better features for image classification and retrieval using Collective Network of Binary Classifiers (CNBC) and feature synthesis
- to develop and experimentally evaluate new methods for feature synthesis

1.2 Thesis Outline

Chapter 2 introduces the basic PSO algorithm along with its extensions. The extensions applied throughout this thesis, MD-PSO and FGBF, are introduced in sections 2.4 and 2.5. At the end of Chapter 2, multiswarm versions of PSO are discussed.

Chapter 3 concentrates on dynamic optimization. A publicly available test bench, Moving Peaks Benchmark (MPB), is introduced along with its multidimensional extension proposed in Publication IV. The chapter continues with a discussion on how to apply multiswarm FGBF and MD-PSO on (MD-)MPB. Finally, the chapter presents some experimental results from Publication IV.

PSC is the topic of Chapter 4. Different particle encoding options are explained and PSC techniques based on Multi-Elitist Particle Swarm Optimization (MEPSO) and MD-PSO with FGBF are introduced in detail. The last section discussing how to evaluate fitness in PSC is based on Publication VIII.

Chapter 5 discusses Artificial Neural Networks and how to train them with MD-PSO. Section 5.1 focuses on Multilayer Perceptrons (MLPs) and Section 5.2 on RBFNNs. For MLPs, Backpropagation (BP) and PSO training methods are compared (Publication I).

For RBFNNs, a novel approach to carry out training using class-specific clustering is introduced (Publication VII).

In Chapter 6, the CNBC classifier is introduced along with its application to Content-based Image Retrieval (CBIR). Chapter 6 is based on Publication II.

Chapter 7 explains the concept of feature synthesis and its objectives. Section 7.2 describes how to use MD-PSO for Evolutionary Feature Synthesis (EFS) as originally proposed in Publication III. An improved fitness evaluation technique designed for MD-PSO-based EFS is also introduced following Publication V. Section 7.3 introduces a different feature synthesis method based on one-against-all perceptrons. This technique was originally proposed in Publication VI.

Chapter 8 concludes the introductory part of the thesis. Publications are provided at the end of the thesis.

1.3 Author's Contribution to Publications

I The publication compares PSO and BP in training MLPs applied for solving medical diagnosis problems. The author contributed to the implementation and the simulations reported in the paper and to the writing of the paper.

II The publication proposes using CNBC for general CBIR. Experiments on benchmark image databases show that with CNBC a significant performance improvement is achieved over traditional retrieval techniques. The author contributed to the formulation of the problem, to the design and implementation of the experiments, and to the writing of the paper.

III The publication proposes using MD-PSO for EFS. In the proposed method, MD-PSO is used to search for optimal arithmetic operators to transform selected elements of original low-level features into more discriminative features. The experiments on an image database show that the synthesized features exhibit an increased discrimination between different classes. The original idea of using MD-PSO for feature synthesis was invented by Prof. Serkan Kiranyaz. The author contributed to the design and implementation of the proposed method and to the writing of the paper.

IV The publication proposes a multiswarm version of FGBF and MD-PSO. This version is beneficial in dynamic optimization, where the search space is slowly changing. Multiple swarms can track multiple optima and detect immediately if a former local optimum becomes the global optimum. The paper also extends MPB to a multidimensional version that allows testing in an environment where also the solution dimensionality is unknown and changing. The author implemented the multiswarm extensions of FGBF and MD-PSO as well as the multidimensional extension of MPB. She carried out the experiments, wrote several sections of the paper, and created figures 3-11.

V An improved version of feature synthesis using MD-PSO is proposed in this publication. In the improved version, a target output vector is defined for each class and MD-PSO is looking for such feature transformations that modify all the feature vectors of a class toward these target vectors. Fitness of a particle is then evaluated as the Mean Squared Error between the obtained and target feature vectors. The

computational complexity of the new fitness evaluation approach is reduced and the obtained results are better than with the approach used in Publication III. The author invented and implemented the novel approach for fitness evaluation. Prior to this publication, the improved fitness evaluation was published in another paper [82]. However, it was originally invented for this work. The author carried out the experiments and wrote the paper.

VI The publication proposes a novel technique for feature synthesis. The technique uses parallel one-against-all perceptrons to generate new features with a higher discrimination power. This in turn leads to improved classification and retrieval results. The main merits of the proposed technique are its simplicity and faster computation compared to existing feature synthesis methods. Extensive simulations show a significant improvement in the features' discrimination power. The author designed the work, made the required implementations, carried out the experiments, and wrote the paper.

VII The publication proposes to train RBFNNs through class-specific clustering (i.e., clustering each class separately to obtain Radial Basis Function centers). This has been done in some earlier works but without considering its impact. It is shown in the paper that the class-specific approach significantly reduces the overall complexity of the clustering. Experimental comparisons against the traditional input and input-output clustering approaches are carried out using the MD-PSO, APC-III, and K-means algorithms. The results show that the class-specific approach can also lead to a significant gain in the classification performance especially when used on networks with relatively few neurons. The author designed the work, made the required implementations, carried out the experiments, and wrote the paper.

VIII The publication proposes a new way to perform fitness evaluations in PSC, namely Fitness Evaluation with Computational Centroids (FECC). An extensive comparison of different fitness functions in PSC is carried out using both MD-PSO and MEPSO. The proposed FECC approach is found to significantly improve clustering results. Different Clustering Validity Indices (CVIs) as fitness functions are ranked and their benefits in different situations are analyzed. The idea of FECC was born is discussions with Kaveh Samiee. The author implemented the compared CVIs, their FECC versions, and some changes to MD-PSO and FGBF. Kaveh Samiee helped checking the correctness of implemented CVIs. The author carried out the experiments and wrote the paper.

2 Particle Swarm Optimization and Extensions

2.1 Stochastic Optimization

When randomness is involved in optimization, the process is called *stochastic optimization*. The randomness can occur in two ways: there is random noise in the fitness evaluations and/or there is randomness in the selection of the search direction. In this thesis, only the latter case is further considered. Stochastic algorithms have become popular during the last few decades in several application areas. They are mainly used in optimization problems which can be highly nonlinear, multimodal, high-dimensional, or otherwise too difficult to be solved using deterministic methods. The problems are often so challenging that it is not feasible to find the global optimum, but the randomness can help stochastic methods in avoiding some local optima and in achieving satisfactory final results. [108]

In 1983, Kirkpatrick et al. proposed Simulated Annealing (SA) [71] that emulates annealing in metallurgy. In annealing, metals are first heated and then slowly cooled to improve their crystal structure and remove defects. This reduces internal stress and makes metals stronger. In SA, the search process is heated by allowing the algorithm to proceed also to solutions having worse fitness values with a certain probability. In the cooling phase, the probability of accepting worse fitness values decreases and, finally, the algorithm only accepts better solution and converges to an optimum. If the optimization problem is difficult, the final solution is still likely a local optimum. Nevertheless, annealing can help the algorithm to escape from some local optima and the final solution is usually significantly improved compared to the deterministic counterpart.

Evolutionary Algorithms (EAs) are stochastic optimization methods using techniques from biological evolution. Well-known evolutionary algorithms are Genetic Algorithm (GA) [41], Genetic Programming (GP) [8], Evolutionary Programming (EP) [36], and Evolutionary Strategies (ES) [36]. GA evolves solutions mimicking operations from genetics: mutation, crossover, and selection. Solutions are typically represented using fixed-length arrays. GP is a similar technique, but uses variable-size tree representation for the candidate solutions. Also the genetic operations used in GA and GP are slightly different. Nevertheless, the difference between the two algorithms is small [121]. Also EP and ES are similar to GA, but they emphasize different parts of natural evolution. GA emphasizes chromosomal operators, EP behavioral changes at the level of the species, and ES behavioral changes at the level of the individual [36]. Extensions of these basic algorithms have made it sometimes hard to distinguish between the methods.

Particle Swarm Optimization (PSO) belongs to the class of swarm intelligence algorithms. They are considered relatives of EAs, but, instead of natural evolution, they mimic the

behavior of animal swarms in their search for food. In addition to PSO, Ant Colony Optimization (ACO) [31] is a widely-used swarm intelligence algorithm. ACO is inspired by foraging ants. Initially, ants search randomly the surroundings of their nest. When they find some food, they deposit chemical pheromones to guide other ants toward the food.

Several comparisons of different stochastic optimization algorithms have been conducted, e.g., [5, 37, 75]. However, most comparisons are restricted to the basic versions of the stochastic optimization methods, while probably hundreds of versions of each base algorithm have been suggested. Furthermore, the comparisons usually concentrate on a very specific application area and the manually selected parameter values may significantly affect the ranking. Not surprisingly, the available comparison results are often contradictory and incomplete. Some more systematic comparison approaches have been suggested (e.g., [91]), but they have not been extensively applied. Also theoretically, it is not possible to find an optimization method that is universally successful in all kinds of problems. The *no free lunch* theorem for optimization [120] basically states that any algorithm which is successful on certain class of optimization problems is guaranteed to perform poorly on another class. Luckily, many real world optimization problems have a similar nature and it is still possible to find algorithms performing well on several problems which are important in practice [53].

2.2 PSO Algorithm

The basic form of the PSO algorithm was introduced in [59] and later modified in [107]. In the algorithm, a swarm of S particles explores a d-dimensional search space, where particle positions are potential solutions to an optimization problem. Initially, each particle, p, has a random position, $\mathbf{x}_p$, and velocity, $\mathbf{v}_p$. At the beginning of each iteration, t, the new particle positions are evaluated using a problem-specific fitness function, $f(\mathbf{x}_p)$. Each particle remembers its personal best solution so far, $\mathbf{b}_p$, and the swarm as a whole remembers the overall best solution globally achieved so far, $\mathbf{b}_S$. If the new particle positions reveal better solutions, $\mathbf{b}_p$ and $\mathbf{b}_S$ are updated. Then the particles are moved to a new position using the following velocity and position update functions:

$$\begin{aligned} \mathbf{v}_p(t+1) &= w(t)\mathbf{v}_p(t) + c_1\mathbf{r}_1(t) \circ (\mathbf{b}_p(t) - \mathbf{x}_p(t)) + c_2\mathbf{r}_2(t) \circ (\mathbf{b}_S(t) - \mathbf{x}_p(t)) \\ \mathbf{x}_p(t+1) &= \mathbf{x}_p(t) + \mathbf{v}_p(t+1), \end{aligned} \tag{2.1}$$

where $w(t)$ is the inertia weight, c_1 and c_2 are the acceleration constants, $\mathbf{r}_1(t) \sim \mathrm{U}(0,1)$ and $\mathbf{r}_2(t) \sim U(0,1)$ are vectors of random variables uniformly distributed between 0 and 1, and $\circ$ denotes a Hadamard (i.e., element-wise) product of vectors. A larger value of $w(t)$ favors exploration, while smaller inertia weight values favor exploitation. Therefore, the inertia weight is often linearly decreased from a high value (e.g., 0.9) to a low value (e.g., 0.4) during iterations of a PSO run as suggested in [107]. For relatively low-dimensional problems (e.g., $d < 50$), the number of particles, S, is usually set higher than the dimensionality, $S > d$. However, it has been observed that the number of iterations, R, required for convergence does not significantly depend on the number of particles. Therefore, it is not meaningful to increase the number of particles when the problem dimensionality increases. Instead, the number of fitness evaluations (i.e., the number of particles multiplied by the number of iterations) should be increased at least linearly with respect to the problem dimensionality [20].

The original PSO is not guaranteed to converge. Instead, the particle velocities have to be controlled by setting a system coefficient, the maximum allowed velocity, $\mathbf{v}_{\max}$, to prevent particle trajectories from swinging out of control. A lot of research has been conducted to study particle trajectories and to find such parameters that guarantee convergence. In [24], Clerc and Kennedy showed that setting the inertia weight as $w = 0.7298$ and the acceleration constants as $c_1 = c_2 = c = 1.4961$ results in convergent behavior. Thereafter, these parameters have been used in most standard PSO versions. The system coefficient $\mathbf{v}_{\max}$ is no longer needed to guarantee convergence. However, it is still often used, set to a very liberal value, in order to achieve more efficient swarm behavior [58].

Several studies have attempted to derive theoretically a parameter region guaranteeing convergence [22, 40, 57, 96, 116], but these studies have made different simplifying assumptions of the PSO process. The assumptions include, e.g., the deterministic assumption, where $\mathbf{r}_1$ and $\mathbf{r}_2$ have fixed values, or the stagnation assumption, where $\mathbf{b}_p$ and $\mathbf{b}_S$ have fixed values. Due to different assumptions, the studies have resulted in different parameter regions. In [96], Poli derived the following inequality guaranteeing convergence using only the stagnation assumption:

$$c < \frac{12(w^2 - 1)}{5w - 7} \tag{2.2}$$

In [23], this parameter region was also empirically tested to strongly correlate with the PSO convergence. However, it was noticed that setting the parameter values near the region edge leads to extremely slow convergence. Thus, the parameter values should be selected from a slightly smaller region.

It should be noted that there are no such parameters that could guarantee the basic PSO to converge to an (local) optimum [117]. The algorithm will simply reach a point of equilibrium if the parameters are properly set. Guaranteed Convergence Particle Swarm Optimization (GCPSO) [115] is a modification of the basic PSO, which has been shown to converge to an optimum. However, GCPSO only improved PSO performance on unimodal problems, especially with small swarms. On multimodal problems, GCPSO did not offer a clear advantage over the basic PSO [117]. Therefore, in general, not having a guaranteed convergence to an optimum is not a problem on more complex problems, where PSO is commonly applied.

Instead, a significant problem with the basic PSO is premature convergence to a local optimum. Even though as a stochastic search algorithm PSO can avoid some local optima, the particles typically gather close to an optimum too early. The global optimum has not been reached yet, but the algorithm loses its ability to explore new areas of the search space. Over the years, numerous modifications of the basic form of the PSO algorithm have been suggested to overcome the problem. Many modifications are based on different swarm topologies. The basic PSO uses *gbest* topology, where each particle receives information of the best solutions found by the entire swarm. In the *lbest* topology, a particle is only connected to its n nearest neighbors. The particle neighborhoods overlap and form a ring topology. In the *lbest* topology, the information flow through the swarm is slower and a newly found global best solution will not affect the whole swarm immediately. This has been repeatedly found to help avoid converging to a local optima [58]. However, Engelbrecht claims in [33] that *gbest* and *lbest* topologies have been previously compared only on very limited datasets. In his extensive comparison, neither approach was clearly better, not even for specific problem classes.

Several PSO variants somehow modify the original $\mathbf{b}_S$ solution to prevent premature convergence: In Multi-Elitist Particle Swarm Optimization (MEPSO) [26], which is better introduced in Section 2.3, $\mathbf{b}_S$ is selected among the $\mathbf{b}_p$ based on their growth rate β_p. In Stochastic Approximation -driven Particle Swarm Optimization (SAD-PSO) [62], stochastic approximation is used to guide the $\mathbf{b}_S$ solution. In Particle Swarm Optimization with an Aging Leader and Challengers (ALC-PSO) [21], other swarm members challenge the current leader when it becomes aged. The lifespan of the leader correlates with its leading power, i.e., its ability to guide the swarm toward better solutions. In Self Regulating Particle Swarm Optimization (SRPSO) [110], a self-regulating inertia weight is used for the $\mathbf{b}_S$ solution. In Fractional Global Best Formation (FGBF) [67], an artificial global best solution is created to compete with $\mathbf{b}_S$. FGBF is introduced in detail in Section 2.5. It is also common to combine PSO with other learning strategies: In Levy Flight Particle Swarm Optimization (LFPSO) [43], Levy flight method is used to redistribute poorly performing particles. In SRPSO [110], the particles use self-perception to select their direction. In Orthogonal Learning Particle Swarm Optimization (OLPSO) [126], Orthogonal Experimental Design (OED) is used to develop a new orthogonal learning strategy that allows PSO particles to use their previous search experience more efficiently. In Particle Swarm Optimization with Multiple Adaptive Methods (PSO-MAM) [47, 48], a non-uniform mutation-based search method, an adaptive subgradient search method, and Cauchy mutation are combined with PSO.

Another problem besides the premature convergence is that the basic form of the PSO algorithm and also nearly all the variants operate in a fixed dimensionality. For clustering, this means that the number of clusters should be known *a priori* or, for neural network training, that the network structure has to be fixed. Similarly, for other optimization tasks in data analysis, the optimal solution dimensionality is usually unknown. An amendment to this problem is introduced in Section 2.4.

Finally, the well-known *curse of dimensionality* problem affects PSO. The size of the search space increases exponentially with respect to the problem dimensionality, d. However, due to computational limitations, it is not possible to similarly increase the number of fitness evaluations. As the number of iterations, R, affects the performance more than the number of particles, S, the number of particles is kept relatively low to allow more iterations. Therefore, for high-dimensional problems, $S \ll d$. S positions in a search space form a $(S-1)$-dimensional hyperplane and the attraction toward $\mathbf{b}_S$ and $\mathbf{b}_p$ makes it difficult to leave the hyperplane. Even though the randomness and the border effects enable escaping the current hyperplane, most of the exploration concentrates on the hyperplane. This increasingly deteriorates the results when the problem dimensionality increases [20]. FGBF can decrease the effect of the *curse of dimensionality* by providing an additional mechanism to leave the current hyperplane.

In the next sections, the PSO extensions applied in this thesis are introduced in detail. MEPSO (Section 2.3) was used in Publication VIII for Particle Swarm Clustering (PSC). Multidimensional Particle Swarm Optimization (MD-PSO) (Section 2.4) and FGBF (Section 2.5) were used in Publications II-VIII for several applications. Multiswarm versions (Section 2.6) were used in Publication IV for dynamic optimization.

2.3 Multi-Elitist Particle Swarm Optimization

In MEPSO [26], each particle has a growth rate β_p. If the fitness value of particle p at iteration t is better than at iteration $t-1$, β_p is increased. The new global best, $\mathbf{b}_S(t)$, is

the particle having the highest growth rate among the particles whose current position has a higher fitness than the previous global best, $\mathbf{b}_S(t-1)$. Thus, the new global best may not have the highest fitness value, but it has improved more during the last iteration. This can help avoid premature convergence, because the global best position will vary more. The pseudocode of MEPSO, as applied in Publication VIII, is given in Listing 2.1. MEPSO is run for R iterations.

Listing 2.1: Pseudocode of MEPSO

```
% Initialization
    1     for p = 1 to S do:
   1.1        randomize x_p(1), v_p(1)
   1.2        initialize b_p(0) = x_p(1)
    2     initialize b_S(0) = x_1(1)
    3     initialize β_p(0) = 0

% MEPSO process
    4     for t = 1 to R do:

        % Updates of growth rate and personal best solutions
   4.1        for p = 1 to S do:
 4.1.1            evaluate the fitness of particle position f(x_p(t))
 4.1.2            if f(x_p(t)) < f(x_p(t−1)) then set β_p(t) = β_p(t−1) + 1
 4.1.3            if f(x_p(t)) < f(b_p(t−1)) then set b_p(t) = x_p(t)
 4.1.4            else set b_p(t) = b_p(t−1)
 4.1.5            if f(x_p(t)) < f(b_S(t−1)) then move particle p to a candidate group B

        % Update global best
   4.2        if |B| > 0 set h to be index the particle with the highest β and set b_S(t) = b_h(t)
   4.3        else set b_S(t) = b_S(t−1)

        % Velocity and position updates
   4.4        for p = 1 to S do:
 4.4.1            compute v_p(t+1) and x_p(t+1) using Eq. (2.1)
```

2.4 Multidimensional Particle Swarm Optimization

MD-PSO [63, 67] is an extension of the basic PSO algorithm providing a solution to the problem of fixed solution dimensionality. MD-PSO particles perform interleaved dimensional and positional PSO processes. The former optimizes the solution dimensionality, while the latter aims at finding the positional optima in particles' current dimensionalities.

For the dimensional process, each particle p knows its current dimensionality, d_p, dimensional velocity, dv_p, and personal best dimensionality so far, db_p. The swarm as a whole keeps track of the overall best dimensionality globally achieved so far, db_S. In the same manner as the regular positional PSO, the dimensional PSO process uses the personal best dimensionality of each particle and the global best dimensionality to attract the particles toward better dimensional solutions.

For each dimensionality, $d \in \{d_{\min}, \ldots, d_{\max}\}$, particles have separate positional PSO components. Positional PSO processes correspond to traditional PSO processes running in a given dimensionality. For the positional processes, the swarm keeps track of the global best position so far achieved, $\mathbf{b}_S^d$, and the particles keep track of their last position,

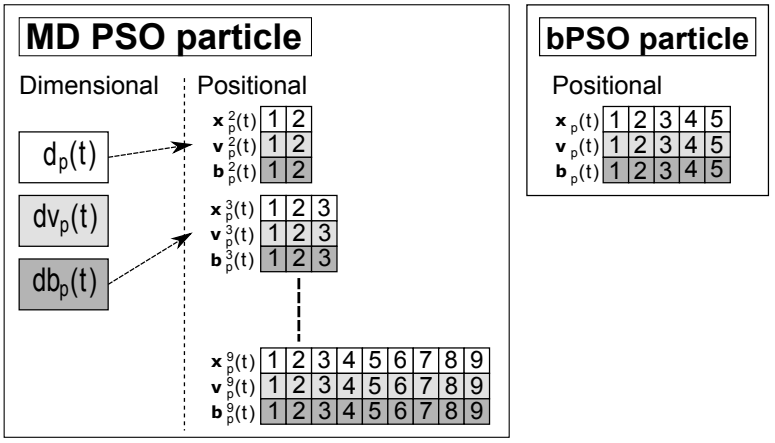


Figure 2.1: MD-PSO particles have dimensional and positional components, while basic PSO (bPSO) particles only have positional components. Here, the dimensional range of MD-PSO is $\{2, \ldots, 9\}$ and the dimensionality of the PSO solutions is fixed to 5. The current dimensionality, d_p, of MD-PSO position is 2 and the personal best dimension, db_p, is 3.

$\mathbf{x}_p^d$, velocity, $\mathbf{v}_p^d$, and personal best position, $\mathbf{b}_p^d$. Thus, the particles can continue their positional search whenever the dimensional updates move them into the particular dimensionality d. The difference of MD-PSO particles and regular PSO particle is illustrated in Fig. 2.1.

As with the basic PSO algorithm, a fitness score for each particle p in its current dimensionality, d_p, is computed at the beginning of MD-PSO iterations. After this, personal best solutions for all particles in their current dimensionalities, $\mathbf{b}_p^{d_p}, \forall p \in \{1, \ldots, S\}$, personal best dimensionalities for all particles, $db_p, \forall p \in \{1, \ldots, S\}$, the global best solution in each dimensionality, $\mathbf{b}_S^d, \forall d \in \{d_{\min}, \ldots, d_{\max}\}$, and the global best dimension, db_S, are updated, if necessary. Then particle positions and velocities are updated in their current dimensionality using (regular) positional PSO updates. The updates are the same as those given in Eq. (2.1), except now the dimensionality of the components depends on the particle's current dimensionality, i.e., the positions and velocities in a particular dimensionality d have d elements.

$$\begin{aligned} \mathbf{v}_p^{d_p}(t+1) &= w(t)\mathbf{v}_p^{d_p}(t) + c_1\mathbf{r}_1(t) \circ \left(\mathbf{b}_p^{d_p}(t) - \mathbf{x}_p^{d_p}(t)\right) + c_2\mathbf{r}_2(t) \circ \left(\mathbf{b}_S^{d_p}(t) - \mathbf{x}_p^{d_p}(t)\right) \\ \mathbf{x}_p^{d_p}(t+1) &= \mathbf{x}_p^{d_p}(t) + \mathbf{v}_p^{d_p}(t+1). \end{aligned} \tag{2.3}$$

After the positional updates, the particle's new position, $\mathbf{x}_p^{d_p}(t+1)$, still has the same dimensionality, $d_p(t)$. Next, the particle goes through dimensional updates, which may move it into another dimensionality. The positional search will continue from $\mathbf{x}_p^{d_p}(t+1)$, if the particle is later moves back to dimensionality $d_p(t)$. Therefore, in all the other dimensionalities, the positional PSO components for iteration $t+1$ are equal to those for iteration t (i.e., $\mathbf{v}_p^d(t+1) = \mathbf{v}_p^d(t)$, $\mathbf{x}_p^d(t+1) = \mathbf{x}_p^d(t)$, $\mathbf{b}_p^d(t+1) = \mathbf{b}_p^d(t), \forall d \in \{d_{\min}, \ldots, d_{\max}\}, d \neq d_p$). The dimensional PSO updates closely resemble the positional ones:

$$\begin{aligned} dv_p(t+1) &= \lfloor dv_p(t) + c_1 r_1(t)\,(db_p(t) - d_p(t)) + c_2 r_2(t)\,(db_S(t) - d_p(t)) \rfloor \\ d_p(t+1) &= d_p(t) + dv_p(t+1), \end{aligned} \tag{2.4}$$

where $\lfloor \cdot \rfloor$ denotes the floor operator. At the end of the MD-PSO algorithm (after R iterations), the global best dimensionality, db_S, and the global best solution in that dimensionality, $\mathbf{b}_S^{db_S}$, represent the optimal dimensionality and solution, respectively. The pseudocode of MD-PSO is given in Listing 2.2 and further details of the algorithm can be found in [63].

Listing 2.2: Pseudocode of MD-PSO

```
% Initialization
1     for p = 1 to S do:
1.1       randomize d_p(1), dv_p(1)
1.2       initialize db_p(0) = d_p(1)
1.3       for d = d_min to d_max do:
1.3.1         randomize x_p^d(1), v_p^d(1)
1.3.2         initialize b_p^d(0) = x_p^d(1)
2     initialize db_S(0) = d_1(1)
3     for d = d_min to d_max do:
3.1       initialize b_S^d(0) = x_1^d(1)

% MD-PSO process
4     for t = 1 to R do:

        % Updates of personal and global best solutions
4.1       for d = d_min to d_max do:
4.1.1         set b_S^d(t) = b_S^d(t-1)
4.1.2         for p = 1 to S do:
4.1.2.1           set b_p^d(t) = b_p^d(t-1)

4.2       for p = 1 to S do:
4.2.1         compute the fitness of particle position, f(x_p^{d_p(t)}(t))
4.2.2         if f(x_p^{d_p(t)}(t)) < f(b_p^{d_p(t)}(t-1)) then set b_p^{d_p(t)}(t) = x_p^{d_p(t)}(t)
4.2.3         if f(x_p^{d_p(t)}(t)) < f(b_p^{db(t-1)}(t-1)) then set db_p(t) = d_p(t)
4.2.4         if f(b_p^{d_p(t)}(t)) < f(b_S^{d_p(t)}(t)) then set b_S^{d_p(t)}(t) = b_p^{d_p(t)}(t)

4.3       db_S(t) = arg min_{d ∈ {d_min, ..., d_max}} (f(b_S^d(t)))

        % Velocity and position updates
4.4       for p = 1 to S do:

            % Regular positional PSO updates
4.4.1         compute v_p^{d_p(t)}(t+1) and x_p^{d_p(t)}(t+1) using Eq. (2.3)
4.4.2         in other dimensions than d_p(t), update v_p^d(t+1) = v_p^d(t) and
              x_p^d(t+1) = x_p^d(t)

            % Dimensional PSO updates
4.4.3         compute dv_p(t+1) and d_p(t+1) using Eq. (2.4)
```

2.5 Fractional Global Best Formation

The basic PSO algorithm, as well as its multidimensional extension MD-PSO, may suffer from premature convergence to a local optimum particularly in high-dimensional multimodal optimization tasks. FGBF [67] is a plug-in to the (MD-)PSO process that can efficiently address the premature convergence problem. FGBF can also decrease the

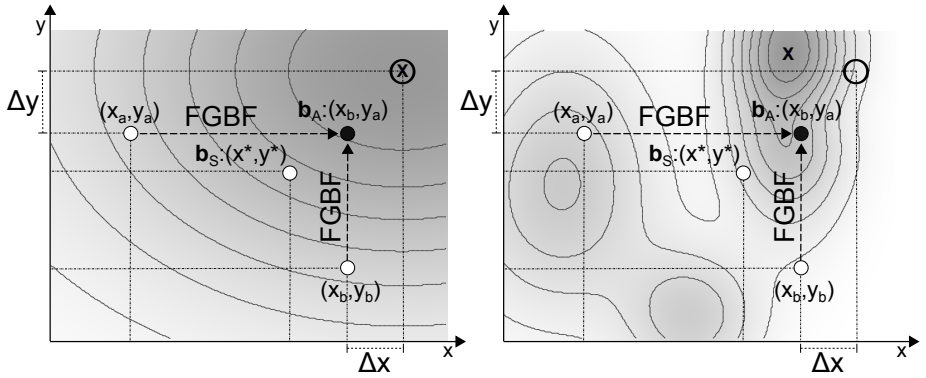


Figure 2.2: FGBF used in a unimodal (left) and a multimodal (right) optimization problem. The overall fitness surface is shown by the background color and the global optimum with $\mathbf{x}$. On both sides, the fractional fitness function is a distance to reference $\mathbf{O}$ and the $\mathbf{b}_A$ solution created by FGBF turns out to be better than the $\mathbf{b}_S$ solution.

effect of the *curse of dimensionality* by providing an additional mechanism to leave the current hyperplane in high-dimensional problems, where $S < d$. The main idea of FGBF is to create an artificial global best solution, $\mathbf{b}_A$, at every iteration by combining the best elements/dimensions of particles' current solutions. This artificial solution then competes with $\mathbf{b}_S$ and the winner with a higher fitness value is then used in the update function Eq. (2.1) or Eq. (2.3). The j^{th} element of a particle solution can be evaluated using a special fractional fitness function, $f_j(\mathbf{x}^d_{p,j})$. In the same manner as the ordinary PSO fitness function, the fractional fitness function must be designed separately for each class of optimization problems and the main challenge is to find such a function that will correlate well with the overall fitness function.

A simplified 2D example of the FGBF process and different fitness functions is shown in Fig. 2.2. On both sides, the fractional fitness score is simply a distance from a reference $\mathbf{O}$ (i.e., $f_j(\mathbf{x}_{p,1}) = \Delta x$ and $f_j(\mathbf{x}_{p,2}) = \Delta y$). Based on this measure, the best 1^{st} element is found to be in particle b and the best 2^{nd} element in particle a, and thus, the $\mathbf{b}^2_A$ solution is created by combining those elements. The overall fitness surface is shown by the background color. Darker color means higher fitness and the global optimum is marked with $\mathbf{x}$. On the left side, the optimization problem is unimodal, and the fractional fitness function perfectly correlates with the overall fitness function. In such cases, the $\mathbf{b}_A$ solution will be always better or at least equally good as the $\mathbf{b}_S$ solution. On the right side, the overall fitness surface is multimodal and it usually means that it is not possible to find a fractional fitness function perfectly matching with the surface. However, even then, the $\mathbf{b}_A$ solution may win the $\mathbf{b}_S$ solution as on the right side of Fig. 2.2. If the $\mathbf{b}_A$ solution turns out to be worse than the $\mathbf{b}_S$ solution, $\mathbf{b}_S$ will be used in the updates as usual.

When FGBF is used together with MD-PSO, a separate artificial solution, $\mathbf{b}^d_A$, is created for every dimensionality d within the dimensionality range $\{d_{\min}, \ldots, d_{\max}\}$. In every dimensionality, the $\mathbf{b}^d_A$ solution competes with the $\mathbf{b}^d_S$ solution. In most optimization tasks, including those discussed in this thesis, it is possible to combine elements from particle positions with different dimensionalities when creating $\mathbf{b}^d_A$. This is illustrated in Fig. 2.3, where the $\mathbf{b}^4_A$ solution with dimensionality 4 is created by combining elements from the current positions of particles a, b, and c with dimensionalities 2, 9, and 3. Two

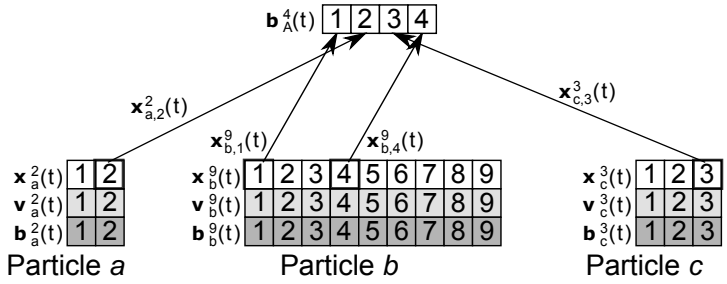


Figure 2.3: FGBF has created an artificial global best solution with dimensionality 4, $\mathbf{b}_A^4$, by combining elements from the current positions of particle a, b, and c. The dimensionalities of $\mathbf{b}_A$, $\mathbf{x}_a$, $\mathbf{x}_b$, and $\mathbf{x}_c$ are all different.

elements are taken from particle b. The figure also illustrates, how only the current solutions with the particle's current dimensionality are considered when creating $\mathbf{b}_A$. The velocities or personal best solutions are not used. $\mathbf{b}_A$ itself is simply a position/solution. It does not need velocity or personal best as it is created separately at every iteration. The pseudocode of FGBF in MD-PSO is given in Listing 2.3 assuming that elements from particle positions with different dimensionalities can be combined. The pseudocode is plugged in between Steps 4.3 and 4.4 in Listing 2.2. Further details of FGBF can be found in [63].

Listing 2.3: Pseudocode of FGBF in MD-PSO

```
% Compute fractional fitness scores
    1     for p = 1 to S do:
   1.1      for j = 1 to d_p(t) do:
  1.1.1       compute f_j(x_{p,j}^{d_p(t)}(t))

% Find the best fractional fitness score for each element j
    2     for j = 1 to d_max do:
   2.1      set p[j] = arg min_{p∈{1,...,S}} (f_j(x_{p,j}^d(t)))

% Create solution b_A in every dimension of the search space
    3     for d = d_min to d_max do:
   3.1      for j = 1 to d do:
  3.1.1       b_{A,j}^d(t) = x_{p[j],j}^d(t)

          % In each dimension compete b_A with b_S
   3.2      if f(b_A^d(t)) < f(b_S^d(t)) then b_S^d(t) = b_A^d(t)

% Update db_S
    4     set db_S(t) = arg min_{d∈{d_min,...,d_max}} (f(b_S^d(t)))
    5     return
```

2.6 Multiswarm PSO

When PSO or its extension is applied, the swarm will eventually converge close to the best solution found by the swarm. In static environments, this can help fine-tune the solution

to reach the exact optimum and it can be considered a desired property if convergence is not premature. In dynamic environments, the locations and heights of the optima can vary and a former local optimum may later become the global optimum. If the whole PSO swarm is gathered around the initially global optimum, it can probably follow moderate changes in the location of this optimum. However, if another optimum becomes the global optimum, the swarm may not detect the change.

Blackwell and Branke [15] addressed this problem by introducing multiswarms. The main idea of multiswarms is that swarms can converge on separate promising optima and together they can establish a comprehensive coverage of the whole search space. The multiswarms are basically separate PSO processes. The particles are only aware of their own process. The only interaction between the swarms is a mutual repulsion preventing two swarms from converging on the same optimum. For this purpose, a repulsion radius, r_{rep}, is defined according to the average radius of the peak basin, r_{bas}. If P peaks are evenly distributed in a d-dimensional space whose extent is X, the repulsion radius can be defined as $r_{rep} = r_{bas} = X/P^{1/d}$. If two swarms move within r_{rep} from each other, the worse performing swarm is simply reinitialized. Physical repulsion is not used, because it could lead to an equilibrium where the mutual repulsion prevents both swarms from approaching an optimum. In an optimal case, the number of multiswarms coincides with the number of optima to be followed. However, in real-life problems the number of optima is usually unknown and may also vary. Therefore, Blackwell suggested also self-adapting multiswarms [12], which can be created or removed during the PSO process.

For a single swarm converging to an optimum, it is essential to maintain enough diversity to be able to follow the optimum despite small changes in its location. Early attempts to maintain required diversity include charged swarms [13] and quantum swarms [15]. In [98] and in Publication IV, we proposed using FGBF as the mechanism to ensure sufficient diversity.

2.6.1 Multiswarms with FGBF and MD-PSO

When multiswarms and FGBF are combined as originally proposed in [98], the multiswarms can track different optima, while FGBF ensures that each swarm can keep tracking an optimum regardless of minor changes in its location. In Publication IV, we proposed combining multiswarms, FGBF, and MD-PSO. This allows us to perform dynamic optimization also in environments where the solution dimensionality is unknown and varying.

We use the above described repulsion radius to reinitialize swarms, if they get too close to each other. We compute the distance between two swarms as the distance of their $\mathbf{b}_S$ positions. To add further diversity, we reinitialize randomly the particle velocities after each environmental change. In the multidimensional case, particle dimensions and dimensional velocities are also reinitialized.

3 Dynamic Optimization

Many real-world datasets have a dynamic nature. Data may be added or removed and data values may be updated. Solving optimization tasks in such constantly changing environments requires specialized techniques. Restarting an optimization algorithm after each system and/or environmental change would lead to a significant loss of useful information when the change is not too drastic. For example, handling video segmentation as a series of static image segmentation tasks without exploiting the similarities between consecutive frames would waste computational resources. Most dynamic optimization problems in machine learning are also multimodal and, therefore, the need of efficient optimization techniques, which are capable of adapting to the changes, is imminent.

EAs designed for dynamic environments started appearing at the end of 1990s, e.g., [3, 6, 17]. Soon many PSO-based approaches were also suggested, e.g., [14, 15, 32, 55, 78]. For dynamic optimization, we proposed in Publication IV to use a combination of multiswarms and FGBF in a fixed dimension and a combination of multiswarms, FGBF, and MD-PSO when the solution dimension is unknown. These algorithms were introduced in Chapter 2. They were tested on Moving Peaks Benchmark (MPB) and its multidimensional extension proposed in Publication IV.

3.1 Moving Peaks Benchmark

MPB [18] is a configurable dynamic environment designed for testing dynamic optimization algorithms in a standard way in a multimodal environment. MPB allows the creation of different dynamic fitness surfaces consisting of a number of peaks with varying locations, heights, and widths. The primary performance measure used is *offline error*, which is the average difference between the optimum and the best solutions found since the last environmental change.

A d-dimensional fitness surface with P peaks is expressed as

$$F(\mathbf{x}, t) = \max\left(B(\mathbf{x}), \max_{q\in\{1,\ldots,P\}} \left(A\left(\mathbf{x}, h_q(t), w_q(t), \mathbf{c}_q(t)\right)\right)\right), \tag{3.1}$$

where $B(\mathbf{x})$ is a time invariant basis landscape, whose utilization is optional, and A is a function defining the height of the q^{th} peak at position $\mathbf{x}$. Each of the P peaks has its own dynamic parameters: height $h_q(t)$, width $w_q(t)$, and position of the peak center $\mathbf{c}_q(t)$.

Each peak parameter can be initialized randomly or set to a certain value and then after every Δt iterations the parameter values are changed. The change over a single peak q at

iteration t can be defined as

$$\begin{aligned} h_q(t) &= h_q(t-\Delta t) + r(t)\Delta h \\ w_q(t) &= w_q(t-\Delta t) + r(t)\Delta w \\ \mathbf{c}_q(t) &= \mathbf{c}_q(t-\Delta t) + \mathbf{s}_q(t), \end{aligned} \tag{3.2}$$

where $r(t) \sim U(0,1)$ is a random variable, Δh and Δw are the height and width change severities, and $\mathbf{s}_q(t)$ is a shift vector, which is a linear combination of a random vector $\mathbf{r}(t)$ and the previous shift vector $\mathbf{s}_q(t-\Delta t)$. The shift vector $\mathbf{s}_q(t)$ is always normalized to length Λ, which is called *change severity*. Accordingly, the shift vector $\mathbf{s}_q(t)$ can be defined as

$$\mathbf{s}_q(t) = \Lambda \frac{(1-\lambda)\mathbf{r}(t) + \lambda \mathbf{s}_q(t-\Delta t)}{||(1-\lambda)\mathbf{r}(t) + \lambda \mathbf{s}_q(t-\Delta t)||}, \tag{3.3}$$

where $||\cdot||$ denotes the Euclidean norm and λ is the correlation factor, which defines the level of randomness of a location change.

The peak type *cone* used in the most experiments is defined with the following equation:

$$A(\mathbf{x}, h_q(t), w_q(t), \mathbf{c}_q(t)) = h_q(t) - w_q(t)\,||\mathbf{x} - \mathbf{c}_q(t)||\,. \tag{3.4}$$

Note that for cone peaks, a higher value of width w_q actually means a narrower cone. 'Slope' could be a better term, but 'width' is used in the original work.

MPB defines three standard settings of parameters, so called *Scenarios*, to allow easier standard comparative evaluations among different algorithms. MPB is no longer available in its original locations, but can be found in [16].

3.1.1 Multidimensional Moving Peaks Benchmark

In the original MPB, the search space dimensionality is fixed *a priori* and it is not varying during the experiment. As discussed earlier, for many real-life optimization problems (e.g., image segmentation), the optimal solution dimensionality is not known. Furthermore, for dynamic optimization problems (e.g., video segmentation) the optimal dimensionality may also be varying. Therefore, in Publication IV, we proposed Multidimensional Moving Peaks Benchmark (MD-MPB) to simulate multidimensional dynamic optimization problems. MD-MPB is a multidimensional extension of MPB. In MD-MPB, the optimal solution dimensionality can vary within a dimensionality range, $\{d_{\min}, \ldots, d_{\max}\}$. Peak positions in different dimensionalities share common elements, which allows exploitation of the information gathered in other search space dimensions.

In MD-MPB, the initialization and changes of peak center positions are carried out only in the highest possible search space dimensionality, $d_{\max}$. Positions in the other dimensionalities are obtained by leaving out the redundant (non-existing) elements. The optimal dimensionality is chosen randomly every time the environment is changed. Therefore, the fitness function with P peaks in a multidimensional environment can be expressed as

$$F(\mathbf{x}^d, t) = \max\left(B(\mathbf{x}^d), \max_{q\in\{1,\ldots,P\}}\left(A\left(\mathbf{x}^d, h_q(t), w_q(t), \mathbf{c}_q^d(t)\right)\right)\right) - (d_{opt} - d)^2, \tag{3.5}$$

where $d \in \{d_{\min}, \ldots, d_{\max}\}$ is the dimensionality of the position $\mathbf{x}^d$, $\mathbf{c}_q^d(t)$ refers to the first d elements of the peak center position, and d_{opt} is the current optimal dimensionality.

In comparison to Eq. (3.1) for unidimensional MPB, Eq. (3.5) has an additional penalty term $(d_{opt} - d)^2$, which is subtracted from the whole environment in the non-optimal dimensionalities.

In MD-MPB, the widths w_q (i.e., the slopes) of cone peaks are scaled by $1/d$, which will make the peaks wider. Thus, the cone peaks are defined as

$$A\left(\mathbf{x}^d, h_q(t), w_q(t), \mathbf{c}_q^d(t)\right) = h_q(t) - w_q(t)\left|\left|\mathbf{x}^d - \mathbf{c}_q^d(t)\right|\right|/d. \tag{3.6}$$

The scaling is done to prevent the benchmark from favoring solutions with lower dimensionalities. Otherwise, a solution whose elements differs from the optimum by 1.0 each, would be a clearly better solution with a lower dimensionality as the Euclidean distance is used.

3.2 Multiswarm FGBF and MD-PSO on (MD-)MPB

3.2.1 Algorithm Adaptation for MPB

To apply multiswarm FGBF on MPB, the PSO fitness function can be set as the complement of MPB fitness function defined by Eq. (3.1), i.e., $f(\mathbf{x}_p(t)) = -F(\mathbf{x}_p, t)$. The fractional fitness function needs to be defined. We used the following fractional fitness function (to be minimized) to evaluate the j^{th} ($j \in \{1, ..., d\}$) element of the position of particle p:

$$f_j\left(\mathbf{x}_{p,j}(t)\right) \;=\; \left(\mathbf{x}_{p,j}(t) - \mathbf{c}_{q,j}(t)\right)^2, \tag{3.7}$$

where $\mathbf{c}_q$ is the centroid of the peak yielding the maximum $A(\mathbf{x}, h_q(t), w_q(t), \mathbf{c}_q(t)))$ either for the position $\mathbf{x}_p$ or, alternatively, for the entire swarm. We call the former mode the *current peak mode* and the latter the *swarm peak mode*. Both modes were considered and evaluated separately. For MD-MPB, the same f_j was used (here given with a notation considering also the solution dimensionality):

$$f_j\left(\mathbf{x}_{p,j}^{d_p(t)}(t)\right) \;=\; \left(\mathbf{x}_{p,j}^{d_p(t)}(t) - \mathbf{c}_{q,j}^{d_p(t)}(t)\right)^2. \tag{3.8}$$

When solution $\mathbf{b}_A$ is created in a certain dimensionality d, we allowed the FGBF algorithm to use elements from particle positions in different dimensionalities. As a matter of fact, as the peaks in MD-MPB directly share the common elements, it is enough to create just one $\mathbf{b}_A$ solution in dimensionality $d_{\max}$ and then the $\mathbf{b}_A$ solutions with lower dimensionalities can be obtained by leaving the excess elements out. To pick the j^{th} element from a solution $\mathbf{x}_p^d$ when creating $\mathbf{b}_A$, its dimensionality naturally has to be j or higher. This approach allows FGBF to exploit the similarity of different dimensionalities and the information gathered in other dimensionalities. Note that it is still possible that, in some search space dimensionalities, $\mathbf{b}_A^d$ beats the current global best solution, $\mathbf{b}_S^d$, while in other dimensionalities it does not.

Due to the shared elements of the peak positions in different dimensionalities in MD-MPB, allowing swarms to converge on the same peak in different dimensionalities would be squandering. Therefore, we extended the mutual repulsion between the swarms to affect swarms in different dimensionalities. To accomplish this, we compute the distance of two swarms still as the distance between their $\mathbf{b}_S$ solutions. If the solutions have different dimensionalities, only the common elements are considered.

3.2.2 Experimental results

In Publication IV, the multiswarm FGBF was tested on MPB Scenario 2 and evaluated against other dynamic optimization methods. The results are given in Table 3.1. The multiswarm FGBF achieved clearly better results than any earlier PSO-based dynamic optimization method. The overall best results were obtained using a method based on the Extremal Optimization algorithm [88], but this algorithm was designed specifically for MPB and its applicability for other practical dynamic problems is not clear. The best FGBF results given in the table were obtained using the swarm peak mode with 10 swarms and 4 particles in each swarm.

Table 3.1: Best results on MPB Scenario 2 [Publication IV]

Source	Base algorithm	Offline error
Blackwell and Branke [14]	PSO	2.16 ± 0.06
Li et. al [78]	PSO	1.93 ± 0.06
Mendes and Mohais [84]	Differential Evolution	1.75 ± 0.03
Blackwell and Branke [15]	PSO	1.75 ± 0.06
Multiswarm FGBF	**PSO**	$\mathbf{1.03 \pm 0.35}$
Moser and Hendtlass [88]	Extremal Optimization	0.66 ± 0.02

When we examined the *current error* (the difference between the current global maximum and the current best result achieved by the algorithm) during the test runs, we observed the expected behavior illustrated in Fig. 3.1. The error temporarily grew larger after each environmental change, but soon the algorithm usually caught the global optimum again. Importantly, at the very beginning, the error was generally clearly larger than immediately after later environmental changes. This shows the benefit of using an algorithm designed for dynamic optimization instead of restarting the algorithm after each environmental change. It was also observed that, during the first few Δt iterations, the current error was generally higher and the global optimum (zero error) was not yet reached. We concluded that at least some of the swarms did not yet converge on a peak. Generally, the initial convergence seems to be more difficult than to keep tracking a peak after environmental changes. In the few cases where the algorithm could not find the global optimum between environmental changes a swarm had most likely lost its followed peak for some reason. Possibly a peak became too narrow and/or low or maybe two peaks were so close to each other that the swarm was reinitialized as a result of repulsion.

We also evaluated the effect of multiswarms and FGBF on the current error by running experiments without multiswarms or without FGBF. In these experiments the swarm peak mode was used. The results are illustrated in Fig. 3.2. Both multiswarms and FGBF proved to be essential for the algorithm performance. Without multiswarms, the single PSO swarm is most likely converging to a single peak and keeps following that peak. The changed center position can be found quickly, but the solution quality is limited by the current fitness of the peak. The multiswarms alone without FGBF performed somewhat better. Immediately after environment changes, the current error was only slightly worse than with FGBF. However, without FGBF, the algorithm could seldom find the global optimum. Either there was no swarm converging to the highest peak or the changed peak center position could not be found before the environment changed again.

On MD-MPB, we could not make comparisons against other methods as they were not

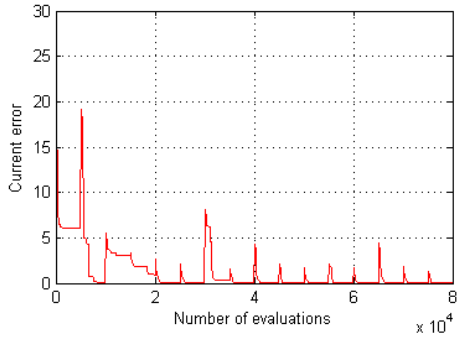
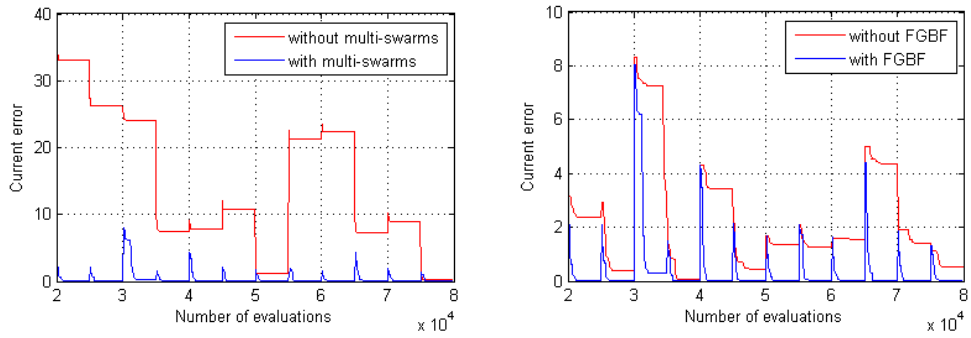


Figure 3.1: Current error for multiswarm FGBF on MPB Scenario 2 [Publication IV]



(a) **(b)**

Figure 3.2: Effect of multiswarms (left) and FGBF (right) on current error on MPB Scenario 2 [Publication IV]

designed to handle multidimensional environments. The best results were now obtained using the swarm peak mode with 10 swarms and 7 particles in each. The combination of multiswarms with MD-PSO and FGBF was able to track the changes of optimal dimensionality usually immediately. This can be largely accredited to FGBF. Also a single swarm with FGBF could effectively track the optimal dimension, but multiswarms without FGBF did not have this ability. Otherwise, the observations on the effect of multiswarms and FGBF were similar to the unidimensional case. Generally, it took a bit longer to overcome the initial phase of a higher current error and to recover from losing peak tracking. This is natural, i.e., as the search space is more complex, the algorithm converges slower.

4 Particle Swarm Clustering

Clustering is an unsupervised process aiming at dividing dataset items into natural groups. Items in one group should be as similar as possible and items in different groups should be well separated. Clustering can be exploited in many more advanced data analysis activities, such as classification, object recognition in images, social network analysis etc. Due to numerous applications benefiting from improved clustering quality, a lot of research activities have focused on different clustering methods and on evaluating the results, i.e., clustering validity assessment.

The most well-known and widely used clustering algorithm is K-means [109]. K-means iteratively finds K centroids and each data item is then assigned to the cluster defined by the closest centroid. Thus, the algorithm always partitions the dataspace into Voronoi cells. While this approach cannot produce partitions with arbitrary shapes (two examples are given in Fig. 4.1), this is usually not a severe problem, since natural data classes commonly have near-Gaussian distributions. Another serious drawback of the K-means algorithm is that it assumes the number of clusters, K, known *a priori*. K-means is also a hill climbing algorithm that will get stuck in the nearest local optimum in the search space. Therefore, when used with complex datasets, the global convergence capability of the K-means algorithm is fully dependent on its initialization. However, K-means is a fast algorithm and can be run several times and using different values of K. For simple datasets, some of the obtained solutions are likely to be sufficiently good and a properly chosen Clustering Validity Index (CVI) can be used to select the best partition and the best cluster number.

Figure 4.1: Two examples of data distributions, which cannot be successfully clustered into Voronoi cells

When the dataset complexity increases, more and more K-means runs are required to find satisfactory partitions until at some point it is probable that all the K-means runs get stuck into local optima far from a satisfactory result. Therefore, with more complex datasets, a more powerful clustering method, such as an evolutionary algorithm [45], deterministic annealing [100], or a swarm intelligence algorithm [1], is needed.

Using PSO for clustering is generally referred as PSC. In PSC, as in other PSO applications, each particle's position represents a complete solution to the problem. This is commonly realized using either partition-based or centroid-based particle encoding. In partition-based encoding [56], each particle is a vector of I integers, where the j^{th} element represents the cluster label assigned to item $j, j \in \{1, .., I\}$ and I is the number of data items to be clustered. In centroid-based encoding, the position of particle p defines a set of potential cluster centroids in an d-dimensional data space: $\mathbf{x}_p = \{\mathbf{m}_{p,1}, \ldots, \mathbf{m}_{p,j}, \ldots, \mathbf{m}_{p,K}\}$, where $\mathbf{m}_{p,j}$ represents the j^{th} cluster centroid and K is the number of clusters. In the same manner as K-means, PSC with the centroid-based encoding divides the dataspace into Voronoi cells. The partition-based encoding can produce clusters of arbitrary shape, but for larger datasets, the particle dimension gets too high, and therefore, the centroid-based encoding is more commonly used.

The first centroid-based PSC technique was proposed in [93]. Its major limitation was the need to manually define the number of clusters, K, *a priori*. Another clustering technique proposed in [94] overcame this limitation by using binary PSO to select which of the potential particle centroids should be included in the final solution, but in this technique the K-means algorithm was used to refine centroid positions.

4.1 Clustering with MEPSO

Das et al. [26] proposed an algorithm using MEPSO (described in Section 2.3) for selecting the number of clusters and finding the centroid positions. They proposed also a particle encoding scheme where each particle position consists of a user-defined maximum number of cluster centroids, $K_{\max}$, along with activation thresholds for each centroid. The position of particle p is encoded as a $K_{\max} + K_{\max} * d$-dimensional vector: $\mathbf{x}_p = \{T_{p,1}, \ldots, T_{p,K_{\max}}, \mathbf{m}_{p,1}, \ldots, \mathbf{m}_{p,K_{\max}}\}$, where $T_{p,j}$, $j \in \{1, \ldots, K_{\max}\}$ is an activation threshold in the range of $[0, 1]$ and $\mathbf{m}_{p,j}$ represents the j^{th} (potential) cluster centroid. The activation thresholds can inactivate cluster centroids, and thus, the final number of clusters can be below $K_{\max}$. The j^{th} centroid is included in the solution proposed by particle p only if $T_{p,j} > 0.5$. If there are less than two active clusters in a solution, one or two randomly selected activation thresholds, $T_{p,j} < 0.5$, are reinitialized to a random value in the range of $]0.5, 1]$. Xu et al. used similar particle encoding for Differential Evolution Particle Swarm Optimization (DEPSO) in [122, 123]. They also deactivated any cluster having less than two items by setting its activation threshold to a random value in the range of $[0, 0.5]$ and then checked whether the condition on the minimum number of clusters is still satisfied. In Publication VIII, this slightly modified approach was applied with MEPSO.

We observed in Publication VIII that the above described particle encoding is susceptible to underclustering, i.e., often finds too few clusters. A probable reason is that each particle position basically represents partitions with all the considered numbers of clusters simultaneously. The algorithm will likely first encounter some decent solutions with smaller numbers of clusters and particle positions will be attracted closer to these solutions. However, in clustering, the same centroid positions are usually not optimal for higher numbers of clusters. This is illustrated in Fig. 4.2. If the algorithm first finds

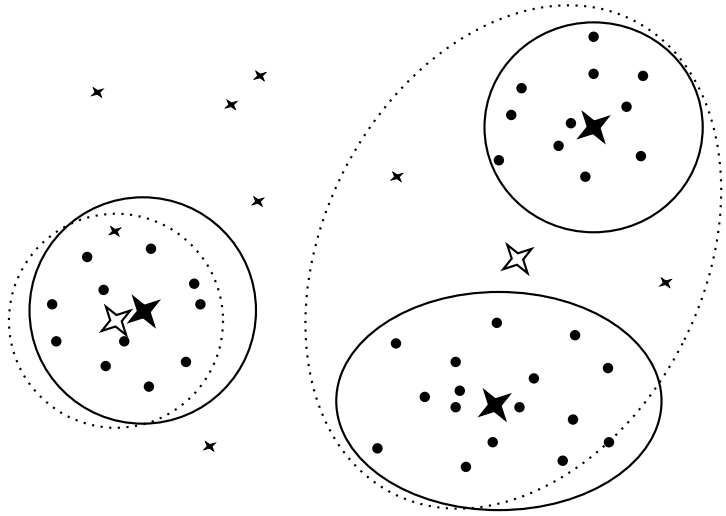


Figure 4.2: Possible cluster centroids (stars) for a dataset. Filled large stars form a desired solution with three clusters and unfilled large stars with two clusters. [Publication VIII]

a near-optimal two-cluster solution shown by unfilled large stars, just adding a third centroid does not form a proper three-cluster solution. This means that the particle encoding creates strong local optima with lower numbers of clusters. Often the final number of clusters is too low because one of such optima has been too difficult for the applied algorithm to escape.

4.2 Clustering with MD-PSO and FGBF

MD-PSO and FGBF were first applied to clustering in [67]. Later, clustering with MD-PSO and FGBF has been applied, for example, for dominant color extraction [69], Holter register classification [64], and Radial Basis Function Neural Network (RBFNN) training ([54, 60], Publication VII). It has been observed that, when adapted properly, MD-PSO with FGBF can lead to an optimal clustering performance even in highly complicated datasets.

A straightforward multidimensional extension of the basic centroid-based encoding can be used with MD-PSO. In the extension, particles can move in every search space dimensionality $d \in \{d_{\min}, \ldots, d_{\max}\}$ and in the d^{th} dimensionality particle positions represent d potential cluster centroids, i.e., $\mathbf{x}_p^d = \{\mathbf{m}_{p,1}, \ldots, \mathbf{m}_{p,j}, \ldots, \mathbf{m}_{p,d}\}$. The dimensional PSO process guides particles toward the optimal number of clusters, while the positional PSO process helps refine the centroid locations with a certain number of clusters. As MD-PSO has separate solutions for different cluster numbers, it will not encounter similar strong local optima with low numbers of clusters as MEPSO (described in Section 4.1) and underclustering is not a problem to the same extent.

FGBF can improve the clustering performance by combining the best individual elements of cluster positions, i.e., cluster centroids. It creates an artificial solution, $\mathbf{b}_A^d$, to compete with the $\mathbf{b}_S^d$ solution in every dimensionality $d \in \{d_{\min}, \ldots, d_{\max}\}$. However, it is challenging to evaluate individual solution elements, because the quality of a certain centroid always depends on the other selected centroids. This is evident in Fig. 4.2.

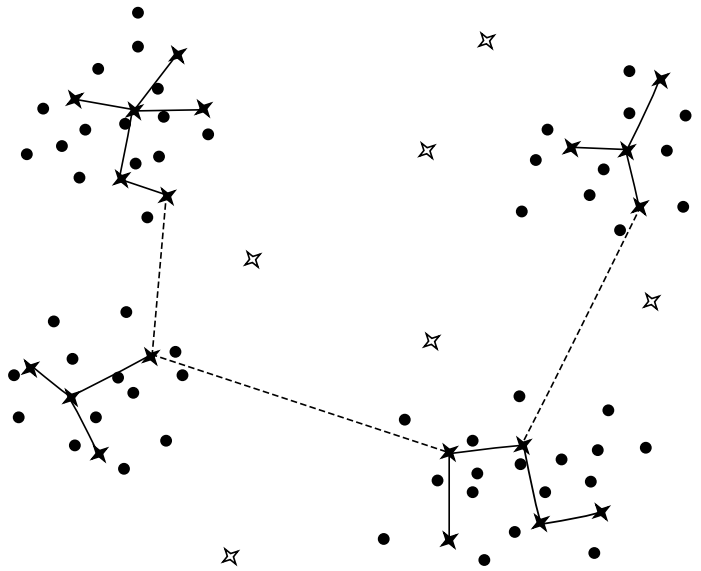


Figure 4.3: To ensure selecting centroids representing different natural clusters, the FGBF operation exploits MSTs

Combining the unfilled large stars to form $\mathbf{b}_A^2$ or the filled large stars to form $\mathbf{b}_A^3$ would probably help MD-PSO find better overall clustering solutions, but combining both filled and unfilled stars into a single solution is clearly not desired. A poorly chosen centroid combination can also result in a situation where no data items are assigned to the clusters defined by certain centroids. It is not feasible to consider all the possible centroid combinations when creating $\mathbf{b}_A^d$ solutions, but purely local centroid fitness evaluation based only on the data items without consideration of other centroids in the solution would likely produce undesired solutions.

To ensure that at least one data item is assigned to each cluster, the FGBF operation uses only centroids which, among all the possible centroids, are closest to at least one data item. To avoid selecting centroids representing the same natural data cluster, spatially close centroids are first grouped into centroid groups using a Minimum Spanning Tree (MST) [74]. A certain number of centroid groups, $d \in \{d_{\min}, \ldots, d_{\max}\}$, can be obtained from the MST simply by breaking the $d-1$ longest MST branches. The artificial solution $\mathbf{b}_A^d$ is then formed by choosing only one centroid form each group. This is illustrated in Fig. 4.3. Potential cluster centroids are represented by stars. The filled stars form the subset of centroids having at least one data item closest to them, while the unfilled stars will not be used in the creation of $\mathbf{b}_A^d$. When creating $\mathbf{b}_A^4$, the $4-1=3$ longest MST branches marked with dashed line would be first broken to obtain 4 centroids group. Only one centroid is then selected from each group.

Originally, in [67] and in most applications thereafter, the fitness function for MD-PSO clustering was set as the quantization error or the average distance of data items to their cluster centroids:

$$f(\mathbf{x}_p^d) = \frac{1}{d}\sum_{j=1}^{d}\sum_{\mathbf{z}\in C_j}\frac{\|\mathbf{z}-\mathbf{m}_{p,j}\|}{I_j}, \tag{4.1}$$

where d is the solution dimensionality (number of clusters), a data item $\mathbf{z}$ belongs to the cluster C_j, if the centroid $\mathbf{m}_{p,j}$ is closer to it than any other centroid in $\mathbf{x}_p^d$, and I_j is the number of data items belonging to C_j. The fractional fitness function to evaluate the individual element $\mathbf{x}_{p,j}^d$ (i.e., centroid $\mathbf{m}_{p,j}$) was simply the average distance between the centroid and the data items belonging to that cluster in the overall solution defined by $\mathbf{x}_p^d$:

$$f_j(\mathbf{x}_{p,j}^d) = \sum_{\mathbf{z} \in C_j} \frac{\|\mathbf{z} - \mathbf{m}_{p,j}\|}{I_j}. \tag{4.2}$$

Naturally, it is not guaranteed or even probable that the same data items will be assigned to the cluster defined by the centroid $\mathbf{x}_{p,j}^d$ when combined with different centroids in solution $\mathbf{b}_A^d$. Nevertheless, these fractional fitness function values are easy to compute simultaneously with computing the overall fitness function values according to Eq. (4.1) and they serve as a proper approximation of the centroid quality.

In Publication VIII, we conducted an extensive comparison of CVIs that can be used for fitness evaluation in PSC. The quantization error was not among the best performers. We also proposed an improved strategy to carry out fitness evaluation. The findings of Publication VIII are discussed in Section 4.3. The fractional fitness score used in Publication VIII was slightly modified from Eq. (4.2) to

$$f_j(\mathbf{x}_{p,j}^d) = \sum_{i=1}^{M} \frac{\|\mathbf{z}_{[i]} - \mathbf{m}_{p,j}\|}{M}, \tag{4.3}$$

where $M = \min(10, I_j)$ and $\mathbf{z}_{[i]}$ represents the i^{th} closest data item to $\mathbf{m}_{p,j}$. Due to computational simplicity, only data items assigned to cluster C_j defined by $\mathbf{m}_{p,j}$ in solution $\mathbf{x}_p^d$ were considered. It should be remembered that the FGBF operation is only used as means to avoid premature convergence. If FGBF cannot improve the results, it will not have any effect on MD-PSO as the artificial solutions will not be used.

To further fine-tune the FGBF operation for clustering, the fractional fitness function could be selected together with the clustering fitness function. This could allow computing the fractional fitness functions values simultaneously with computing the overall particle fitness as in the earlier works where Eq. (4.1) and Eq. (4.2) were used as the overall and fractional fitness functions, respectively. In addition to the computational benefit, further resemblance of the overall and fractional fitness functions might also result in improved $\mathbf{b}_A$ solutions, but this remains a topic for future research.

4.3 Evaluation of Clustering Validity in Particle Swarm Clustering

The quality of a clustering result, i.e., a data partition, can be evaluated using a CVI. External CVIs compare the partitions with some ground truth information such as item labels. In general, there is no need for clustering if such ground truth data is available, and therefore, external CVIs are mainly used for verification purposes. Internal CVIs evaluate partitions purely based on some internal data properties such as cluster compactness and separation. Relative CVIs allow ranking of different partitions. PSC methods generally select an internal relative CVI as their fitness functions. The selection of a proper CVI is essential to obtain proper results. If the selected fitness function does not model the search space well, the clustering results will be poor no matter how advanced the applied

algorithm is. Nevertheless, fitness function selection in PSC has received surprisingly little attention. Xu et al. [123] provide some insight into selecting the PSC fitness function, but most new PSC applications simply adopt a fitness function from an earlier work, while often changing it would be an easy way to improve the results. There are several general CVI comparisons, e.g., [4, 29, 86, 118], but their results are not consistent and may not directly correlate with the suitability of the CVIs to be fitness functions in dynamic PSC.

Because in dynamic clustering the number of clusters is unknown, it is important to know how a specific CVI will behave with varying cluster numbers. Several CVIs which have been successful in general CVI comparisons do not get their minimum/maximum value on the optimal number of clusters. With them, different statistics, such as maximum increase/decrease or maximum/minimum of the second differences, are better suited to detect the optimal number of clusters [29, 38]. Such CVIs as fitness functions may mislead PSO, which is simply searching for the optima. It should also be noted that PSC is often a pre-step to a more advanced operation, such as Electrocardiogram (ECG) classification in [64] or training of RBFNNs in Publication VII. Each obtained cluster is processed in a certain manner, and commonly, it is easy to process few clusters similarly, but it is more problematic if the items assigned to a single cluster should not have the same treatment. Therefore, if the partitions are otherwise meaningful, CVIs causing the PSC application to overcluster can be usually preferred to those CVIs that will more likely lead to underclustering.

Most existing CVIs depend on the cluster centroids in some way. In PSC fitness evaluation, the centroid locations in CVI formulas are traditionally directly replaced by the particle positions [67, 93, 122]. This seems appropriate, as the purpose of the fitness function is to evaluate the fitness of the particle position as a clustering solution. However, usually the computational centroid of the items assigned to the corresponding cluster differs from the exact position defined by the particle as illustrated in Fig. 4.4. It can be argued that, if two different sets of centroids result in the same final data partition, their fitness should be equal. Furthermore, in a situation similar to that in Fig. 4.4, the traditional PSC approach would most probably later converge so that particle positions and computational centroids match, but the computational effort required would be wasted in the sense that the final data partition would not improve further.

In Publication VIII, we proposed using Fitness Evaluation with Computational Centroids (FECC), where the fitness of a particle position is computed using the values of the corresponding computational centroids. With FECC, the particle positions resulting in the same final partition will get the same fitness value. However, in real-life clustering problems, clusters are usually not as well separated as in Fig. 4.4. Moving the particle centroids to match with the computational centroids would result in different cluster memberships, which is a well-known fact from the functioning principle of K-means. We believe that in such situations FECC has a greater potential to guide the PSC process toward better partitions. With the traditional approach, a promising partition may be penalized and lost only because the particle positions are somewhat outside the data distribution. The FECC approach can be easily used with any PSC method where the fitness evaluation is performed using a CVI depending on the cluster centroids. The only modification needed is the computation of the mean of the data items assigned to a certain cluster before computing the value of the selected CVI.

In Publication VIII, we also conducted an extensive comparison of CVIs as PSC fitness functions. We considered 17 CVIs. 14 of them could be applied using both the traditional and FECC approaches. The experiments were conducted over 720 synthetic and 20 real

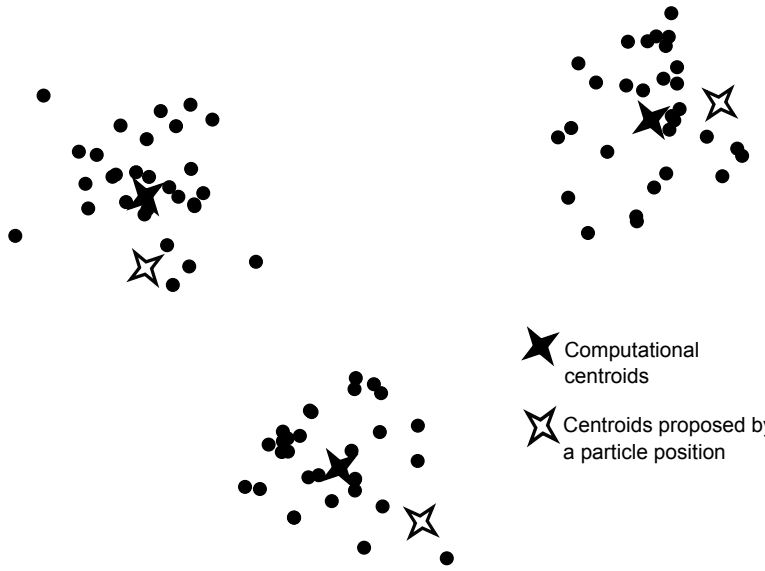


Figure 4.4: An example of different centroids proposed by a particle and computational centroids [Publication VIII]

datasets with varying properties. Both MEPSO and MD-PSO were used for clustering and three different external CVIs were used to evaluate the obtained partitions.

The overall fitness function ranking over the synthetic datasets is given in Fig. 4.5. The fitness functions operating with the FECC approach are denoted with an asterisk. All the best ranking indices were using FECC. The conducted statistical significance analysis confirmed that, for all the best performing CVIs, the FECC approach indeed improved the performance significantly. The winner fitness function was Xu index defined as

$$Xu = d \log\left(\sqrt{\frac{ssw}{dI^2}}\right) + \log K, \tag{4.4}$$

where I is the number of data items, d is data dimension, K is number of clusters, and ssw is the within-cluster sum-of-squares defined as

$$ssw = \sum_{i=1}^{K} ssw_i = \sum_{i=1}^{K} \sum_{\mathbf{z} \in C_i} \|\mathbf{z} - \mathbf{m}_i\|^2. \tag{4.5}$$

Here $\mathbf{m}_i$ is the centroid (the computational centroid for Xu* and the centroid defined by a particle position for Xu) of cluster C_i. The definitions of the other evaluated CVIs are given in Publication VIII. The publication compares the fitness functions under various conditions and provides further recommendations on fitness function selection for different situations.

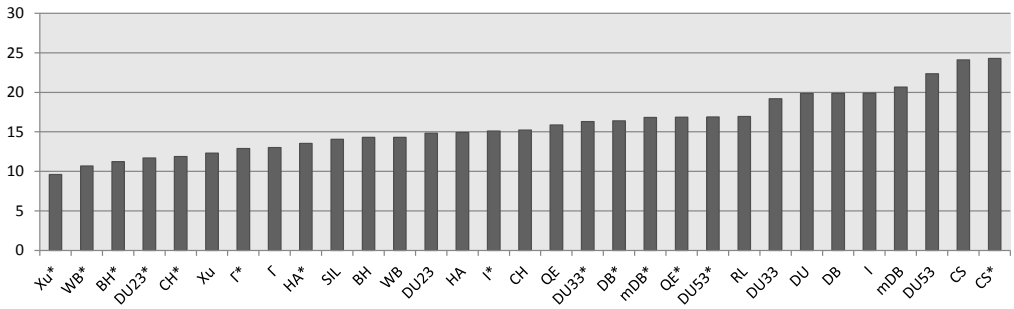


Figure 4.5: Average fitness function ranks for PSC over 720 synthetic datasets [Publication VIII]

5 Training of Artificial Neural Networks

The human brain has an amazing ability to learn from experience and to gain deeper understanding of why things happen the way they do. Artificial Neural Networks (ANNs) have been inspired by the brain and other biological nervous systems. They strive for the same learning and understanding ability and the models have some similarities with biological nervous systems. The purpose has not been to build a faithful model, but to gain a maximal learning ability with feasible computational and memory requirements. Generally, all ANN models are defined using the following three properties [81]:

- *Architecture* defines the network topology and parameters.
- *Activity rules* define how a neuron responds to its inputs.
- *Learning rules* specify how the network parameters change with time.

ANNs can be described as directed graphs whose nodes (i.e., neurons) perform some activation function to their inputs and then give the results forward to be one of the inputs for another node until the output neurons are reached. ANNs can be divided into feedforward and recurrent networks based on their connectivity. The recurrent topology can contain backward loops, while the feedforward topology has only forward connections. The feedforward networks are significantly more common due to their easier training. Usually, they are organized into layers of neurons and only the neurons in adjacent layers can be connected. The input layer is commonly just a passive layer, where no computations are carried out. Therefore, it is not counted to the total number of layers, L. The layers between the input and output layers are called *hidden layers*. A typical activation function is of the form

$$y_k^l = a\left(\sum_{j=1}^{N} \omega_{jk}^l z_j - \theta_k^l\right), \tag{5.1}$$

where y_k^l is the output of neuron k in layer l, N is the number of inputs, z_j is the j^{th} input to the neuron, ω_{jk}^l is the weight for the connection between neuron k and its j^{th} input, and θ_k^l is the bias of the neuron k in layer l. The number of input neurons, N^i, and the number of output neurons, N^o, are defined by the problem, while the number of hidden layers and the number of neurons in each hidden layer must be somehow decided during the network design. A sample feedforward ANN is illustrated in Fig. 5.1. It has three layers: two hidden layers and the output layer. The figure also shows the connection weights w_{j1}^2 and the bias θ_1^2 for the first neuron in layer 2.

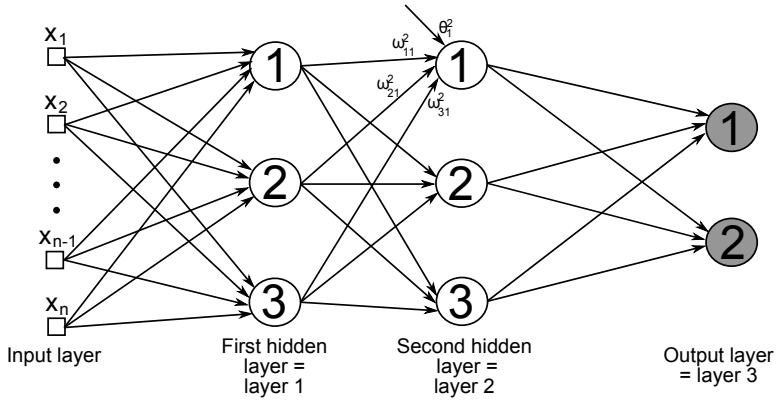


Figure 5.1: An example of fully-connected feedforward ANN [Publication I]

Typical applications for neural networks are classification, regression, data processing (e.g., clustering, filtering), and robotics. Some commonly known network models are Multilayer Perceptrons (MLPs), RBFNNs, Recurrent Neural Networks (RNNs) [89], Autoencoders [9], and Convolutional Neural Networks (CNNs) [73]. In this thesis, MLPs and RBFNNs applied to classification are covered in more detail.

5.1 Multilayer Perceptrons

MLPs are feedforward networks having a topology similar to Fig. 5.1. They have smoothly nonlinear (differentiable everywhere) activation functions. The nonlinearity is important, because otherwise MLPs could always be reduced to Single-layer Perceptrons (SLPs) having equal capabilities. A commonly used activation function is the *sigmoid* function:

$$y_k^l = \frac{1}{1 + e^{-\sum_{j=1}^{N^{l-1}} \omega_{jk}^l y_j^{l-1} - \theta_k^l}}, \tag{5.2}$$

where N^{l-1} is the number of neurons in the previous layer $l-1$. When computing outputs of the first hidden layer, y_j^{l-1} refers to the j^{th} element of the network input, z_j.

5.1.1 Training MLPs with Backpropagation

The most common approach to MLP training is Backpropagation (BP). The early versions of the algorithm date back to 1970s. Rumelhart et al. [101] significantly contributed to the popularization of BP in 1986. During 1990s and 2000s several improvements to the BP algorithm were proposed, but only in 2010s BP has gained its current huge popularity thanks to the emerging deep learning techniques [102].

The BP algorithm can be considered to be a generalization of the least mean squares method. It is an iterative gradient descent technique aiming to minimize the Mean Squared Error (MSE) between the target outputs and actual computed outputs. The BP algorithm consists of two passes: during the forward pass, the input pattern is propagated through the network, and during the backward pass, the error is propagated backwards through the network to update the weights. The general idea of the BP algorithm can be summarized as follows:

1. Initialize randomly all the network weights, ω_{jk}^l, and biases, θ_k^l.
2. Feed a training pattern p to the network. Compute the outputs, y_{kp}^l, for all the neurons.
3. Calculate the error between the computed output, y_{kp}^o, and the desired output, t_{kp}, as $e_{kp} = t_{kp} - y_{kp}^o$ for each output layer neuron k.
4. For each neuron k and for each network parameter h_k (either ω_{jk}^l or θ_k^l) calculate the partial derivatives $\frac{\partial E_p}{\partial h_k}$, where E_p is the total training error computed over all N^o output neurons when training pattern p is fed to the network defined as $E_p = \frac{1}{2}\sum_{k=1}^{N^o} e_{kp}^2$. The name of the BP algorithm comes from this step: for MLPs it is necessary to start calculating the partial derivatives from the output layer and then iteratively proceed backwards toward the input layer. The formulas for the partial derivatives can be found in [44].
5. Update the parameters as

$$h_k[t+1] = h_k[t] - \eta \frac{\partial E_p}{\partial h_k}, \tag{5.3}$$

   where η is the learning rate.
6. Repeat steps 2-5 until a stopping criterion is reached. To reduce the risk of being trapped into a local minimum, it is recommended to randomize the order of the training samples at every *epoch* (full presentation of the whole training set).

The BP algorithm is computationally efficient training algorithm, which explains its current popularity. It can handle huge datasets in a reasonable time. However, it has also some disadvantages. It is, after all, just a gradient descent method, which can get trapped into a local optimum and is dependent on its initialization. The learning rate parameter, η, is very important. Failing to set it properly may lead to either oscillations or extremely long training times. Furthermore, BP can operate in a fixed network architecture only. This means that the architecture must be set prior to training. Usually the architecture is set based on general rules or previous experiments and, therefore, may not be optimal for the current problem.

5.1.2 Training MLPs with Particle Swarm Optimization

To use PSO to train MLPs, the problem must be formulated as a solution space suitable for PSO search. The particle positions can be formed as

$$\mathbf{x}_p = \{\{\omega_{jk}^1\}, \{\theta_k^1\}, \{\omega_{jk}^2\}, \{\theta_k^2\}, \ldots, \{\omega_{jk}^L\}, \{\theta_k^L\}\}, \tag{5.4}$$

where $\{\omega_{jk}^l\}$ and $\{\theta_k^l\}$ are sets of weights and biases needed for layer l.

The quality of the obtained networks can be evaluated using MSE, which is now computed over all I training patterns.

$$MSE = \frac{1}{2IN^o}\sum_{p=1}^{I}\sum_{k=1}^{N^o}(t_{kp} - y_{kp}^o)^2. \tag{5.5}$$

The MSE is directly used as the fitness function for PSO particles.

5.1.3 Comparison of BP and PSO for Training MLPs

Early comparisons of BP and PSO include [83, 114]. In [114], different PSO-based training techniques were shown to be superior to BP and GA-based training in terms of the obtained classification accuracy in the test set. The results obtained by Mazurowski et al. [83] were quite the opposite and the BP training was superior on imbalanced classes. Also in other studies comparing BP with evolutionary training methods such as GA, contradictory conclusions are drawn. We believe that the contradictory conclusions can be explained by the fact that in the earlier comparisons only one or few MLP topologies have been considered. However, the topology selection should not be neglected. It can drastically change the results and for any given problem it may be possible to find a topology where either of the compared algorithm performs better.

In Publication I, we proposed a new comparison procedure that systematically and exhaustively considered all the network architectures within an architecture space defined by the range of layers, $\{L_{\min}, L_{\max}\}$, and the range of numbers of neurons in each hidden layer l, $\{N^l_{\min}, N^l_{\max}\}$. The creation of the architecture space is defined in more detail in Section 5.1.4 and the architecture space used in Publication I is shown in Table 5.2. Furthermore, both algorithms were applied using two different iteration numbers (high and low).

The most important result in Publication I was that the comparison results indeed varied between the different architectures and the iteration numbers. The variation over different architectures is illustrated in Fig. 5.2. It is clear that in the comparisons and the applications, it is not enough to consider a single architecture. Table 5.1 shows the best average test classification errors obtained on different datasets and whether it was obtained using the low or the high iteration number. As can be seen, the iteration number is also an important factor that can change results. In Publication I, PSO demonstrated better average test classification error.

Table 5.1: The best average test classification errors for BP and PSO [Publication I]

Dataset	Method	Test CE	It. number
Breast cancer	BP	0.0101 ± 0.0024	low
	PSO	$\mathbf{0.0078 \pm 0.0024}$	low
Diabetes	BP	0.2175 ± 0.0112	low
	PSO	$\mathbf{0.2135 \pm 0.0024}$	high
Heart disease	BP	0.2222 ± 0.0087	low
	PSO	$\mathbf{0.2043 \pm 0.0031}$	low

5.1.4 Evolving MLPs with Multidimensional Particle Swarm Optimization

MD-PSO was suggested for evolving MLPs in [66]. The dimensional search process of MD-PSO can be dedicated to search for an optimal network topology. This is realized by proposing a systematic way to enumerate all the architectures in the architecture space defined by two given range arrays, $R_{\min} = \{N^i, N^1_{\min}, \ldots, N^{L_{\max}-1}_{\min}, N^o\}$ and $R_{\max} = \{N^i, N^1_{\max}, \ldots, N^{L_{\max}-1}_{\max}, N^o\}$. The numbers of neurons in the input and output layers, N^i and N^o are defined by the problem at hand and, thus, are the same in both arrays. All the architectures are enumerated into hash indices using a hash function. The

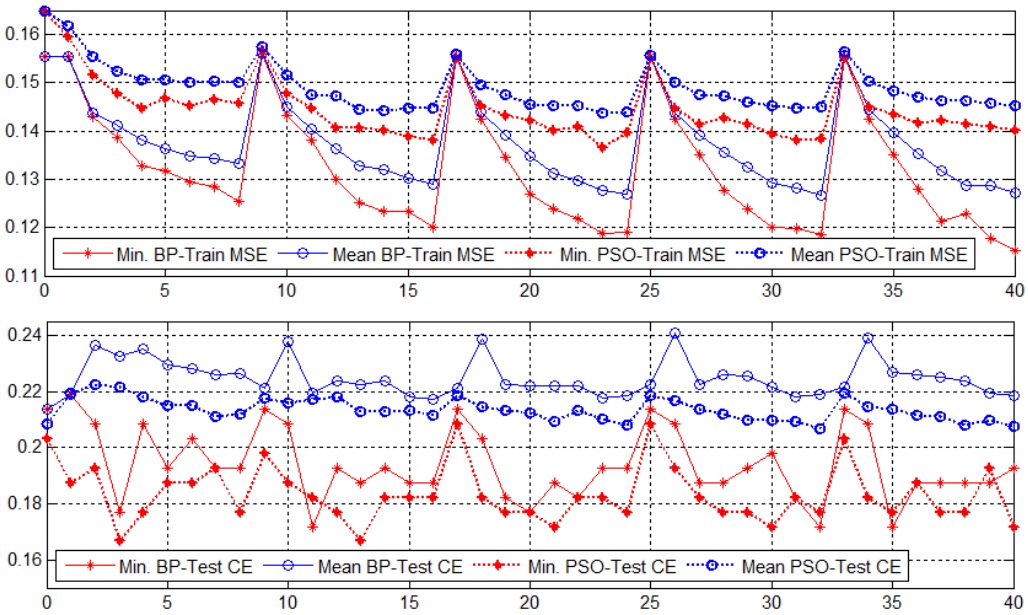


Figure 5.2: Train (top) and test (bottom) error statistics for BP and PSO on the architectures 0-40 defined in Table 5.2 over Diabetes dataset [Publication I]

hash function starts from the simplest network, either a SLP or a MLP with $L_{\min} - 1$ hidden layer and $N^l_{\min}$ neurons in each layer, and continues to the most complex MLP having $L_{\max} - 1$ hidden layers and $N^l_{\max}$ neurons in each layer.

The hash table for sample range arrays $R_{\min} = \{N^i, 1, 1, N^o\}$ and $R_{\max} = \{N^i, 8, 4, N^o\}$ is given in Table 5.2. The hash function associates the first index ($d = 0$) with the simplest considered architecture (SLP) and the most complex architecture receives the highest index ($d = 40$). MD-PSO can then perform its dimensional search by navigating between the network architectures according to the hash table.

For each network topology, MD-PSO performs the regular positional search. The particle encoding is a straightforward multidimensional extension of the encoding in Eq. (5.4):

$$\mathbf{x}^d_p = \{\{\omega^1_{jk}\}, \{\theta^1_k\}, \{\omega^2_{jk}\}, \{\theta^2_k\}, \ldots, \{\omega^d_{jk}\}, \{\theta^d_k\}\}. \tag{5.6}$$

In addition to the optimal solution found in the optimal dimension, db_S, MD-PSO provides optimal solutions in all the other considered architectures. In dynamic systems, where the solution space changes over time, MD-PSO can exploit the other optimal solutions with different architectures when it continues its search in an altered solution space. In this way, the MD-PSO can easily fine-tune the solutions in other architectures as well and it may help it to find better solutions. In this thesis, this property is exploited in Collective Network of Binary Classifiers (CNBC) discussed in Chapter 6. MLPs trained with MD-PSO are also used for feature synthesis as described in Section 7.3.

Fig. 5.3, originally published in [66], illustrates the ability of MD-PSO to find optimal network architecture. In the upper part, the statistics for BP training over all the architectures in the architecture space in Table 5.2 are given. The lower part, shows a histogram of the final best architectures found by 100 MD-PSO evolutions. Clearly, the

Table 5.2: Architecture space defined by range arrays $R_{\min} = \{N^i, 1, 1, N^o\}$ and $R_{\max} = \{N^i, 8, 4, N^o\}$

Index	Configuration	Index	Configuration	Index	Configuration
0	$N^i \times N^o$	14	$N^i \times 6 \times 1 \times N^o$	28	$N^i \times 4 \times 3 \times N^o$
1	$N^i \times 1 \times N^o$	15	$N^i \times 7 \times 1 \times N^o$	29	$N^i \times 5 \times 3 \times N^o$
2	$N^i \times 2 \times N^o$	16	$N^i \times 8 \times 1 \times N^o$	30	$N^i \times 6 \times 3 \times N^o$
3	$N^i \times 3 \times N^o$	17	$N^i \times 1 \times 2 \times N^o$	31	$N^i \times 7 \times 3 \times N^o$
4	$N^i \times 4 \times N^o$	18	$N^i \times 2 \times 2 \times N^o$	32	$N^i \times 8 \times 3 \times N^o$
5	$N^i \times 5 \times N^o$	19	$N^i \times 3 \times 2 \times N^o$	33	$N^i \times 1 \times 4 \times N^o$
6	$N^i \times 6 \times N^o$	20	$N^i \times 4 \times 2 \times N^o$	34	$N^i \times 2 \times 4 \times N^o$
7	$N^i \times 7 \times N^o$	21	$N^i \times 5 \times 2 \times N^o$	35	$N^i \times 3 \times 4 \times N^o$
8	$N^i \times 8 \times N^o$	22	$N^i \times 6 \times 2 \times N^o$	36	$N^i \times 4 \times 4 \times N^o$
9	$N^i \times 1 \times 1 \times N^o$	23	$N^i \times 7 \times 2 \times N^o$	37	$N^i \times 5 \times 4 \times N^o$
10	$N^i \times 2 \times 1 \times N^o$	24	$N^i \times 8 \times 2 \times N^o$	38	$N^i \times 6 \times 4 \times N^o$
11	$N^i \times 3 \times 1 \times N^o$	25	$N^i \times 1 \times 3 \times N^o$	39	$N^i \times 7 \times 4 \times N^o$
12	$N^i \times 4 \times 1 \times N^o$	26	$N^i \times 2 \times 3 \times N^o$	40	$N^i \times 8 \times 4 \times N^o$
13	$N^i \times 5 \times 1 \times N^o$	27	$N^i \times 3 \times 3 \times N^o$		

chosen architectures match with the best BP statistics. Simpler architectures can be seen somewhat favored.

5.2 Radial Basis Function Neural Networks

RBFNNs have been successfully applied in several areas such as medical diagnostics [2], robotics [79], dynamic system design [124], and stock index forecasting [104]. They are feedforward neural networks with universal approximation ability. They always have a single hidden layer of Radial Basis Function (RBF) neurons and a linear output layer. The RBF neurons use a strictly positive radially symmetric activation function to transform the input data to a new feature space usually having a higher dimensionality than the input space. The mathematical justification behind this is *Cover's theorem on the separability of patterns* [25, see 44 p. 258] stating: "A complex pattern-classification problem cast in a high dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space". The number of hidden RBF units is usually high due to the same theorem. The output layer then supplies the network's response to the transformed input data.

RBFNNs commonly have Gaussian basis functions as their activation functions:

$$g(\mathbf{z}) = e^{\left(\frac{-||\mathbf{z}-\mathbf{m}||^2}{2\sigma^2}\right)}, \tag{5.7}$$

where $\mathbf{m}$ and σ are the center location and width of a Gaussian neuron and $\mathbf{z}$ is an input vector. The outputs of RBFNNs are weighted sums of all the Gaussian neuron outputs:

$$y_j = \sum_{k=1}^{N} \omega_{kj} g_k(\mathbf{z}) = \sum_{k=1}^{N} w_{kj} e^{\left(\frac{-||\mathbf{z}-\mathbf{m}_k||^2}{2\sigma_k^2}\right)}, \tag{5.8}$$

where N is the number of Gaussian neurons and w_{kj} is the weight between the k^{th} Gaussian neuron and the j^{th} output neuron.

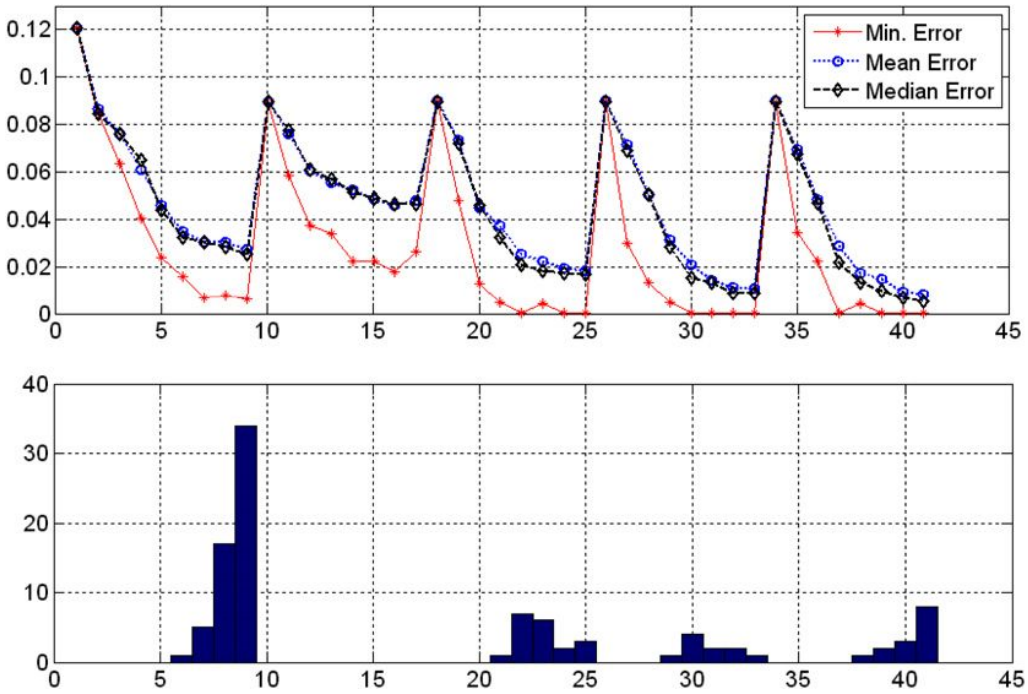


Figure 5.3: Training MSE for exhaustive BP training on architectures 1-41 (top) and architecture histogram of 100 MD-PSO evolutions (bottom) over Heart Disease dataset. Originally published in [66].

5.2.1 RBFNN Training

The training of RBFNNs consists of deciding the number of hidden neurons, assigning the values of their center locations and widths, and finally assigning the weight values. The center locations have been observed to have a more critical impact on the network performance than their widths or the weights [119]. Also the number of hidden neurons is important: too few neurons may be insufficient to learn a model, while too many neurons may lead to overfitting [11, 35].

It is also possible to train RBFNN using BP or MD-PSO as described in sections 5.1.1 and 5.1.4 for MLP. A limitation with BP is that it cannot evolve the network architecture, but it must be fixed *a priori*. Simultaneous search for the network topology and its parameters can be performed using MD-PSO or some evolutionary or genetic algorithms, but these methods are generally slow. A more common approach is to first define the center locations and widths and then determine weights separately in the second phase. Possible techniques for solving the center positions include supervised vector quantization algorithms or supervised training of decision trees [103]. However, the most common approach is to cluster the input items and put the RBF centers to the found cluster centroids.

Originally, clustering was based on fully unsupervised input clustering approach, but soon some partially supervised techniques were introduced. In the commonly used input-output clustering approach, the input vectors are concatenated with the target output vectors, and clustering is performed using these concatenated vectors. In Publication VII, we

proposed a novel approach to bring supervision into the clustering step. We simply clustered items class-by-class. The approach is called *class-specific clustering*.

In addition to RBF center locations, their widths have to be determined. Often acceptable classification results can be obtained even by using the same widths for all centroids [44]. However, using a single width value may harm performance if there is a reason to assume that RBF centroids correspond to clusters with varying sizes. This is typical for the class-specific clustering approach as the distributions of different classes may be completely different. Varying width values for each RBF neuron can be obtained using a simple heuristic [52, 87] where, for each center, $\mathbf{m}_k$, the distance to the nearest RBF center is computed and multiplied by γ:

$$\sigma_k = \gamma \min_{k \neq j} (\|\mathbf{m}_k - \mathbf{m}_j\|). \tag{5.9}$$

With the class-specific clustering, only the clusters belonging to the specific class can be considered when computing the distance. In [52], $\gamma = 5$ was applied.

In the second learning phase, the network weights are determined typically using either BP [87, 90] or the linear least squares technique [44]. As the only parameters to be set are the weights between the RBF and output neurons, BP reduces to a linear regression task.

5.2.2 Different Clustering Approaches in RBFNN Training

5.2.2.1 Input Clustering

The input clustering refers to the basic unsupervised clustering, where the available class information is not considered in any way. The input clustering has been widely used in RBFNN training, e.g., in [42, 44, 92].

5.2.2.2 Input-output Clustering

In the input-output clustering, the vectors to be clustered, $\mathbf{z}_p^{IO}$, are formed by concatenating the input vectors and the corresponding target output vectors weighted by a factor β:

$$\mathbf{z}_p^{IO} = [\mathbf{z}_p, \beta \mathbf{t}_p], \tag{5.10}$$

where $\mathbf{z}_p$ is the p^{th} input vector in the training set and $\mathbf{t}_p$ is the corresponding target output vector. The weight β is commonly selected so that each element in the augmented vector will have a similar order of magnitude [19]. As a result, the output part will be emphasized more when the number of classes is higher and, thus, the 1-of-C-encoded output vector (e.g., [1 -1 -1], [-1 1 -1], and [-1 -1 1] for the classes of a 3-class problem) longer. After the input-output clustering, the RBF centers are obtained from the cluster centroids by discarding the output part. The final network will operate with the inputs only, as usual. It has been shown that with a suitable weight β the training error can be made arbitrarily small by choosing a sufficiently large number of hidden neurons [112]. However, this may lead to poor generalization ability and to an undesired network structure.

While the input-output clustering is assumed to be superior to the input clustering, only few studies have actually compared the two approaches. In [19], a better performance as a dynamic model for a time series system was obtained when using the input-output clustering approach, but in that comparison also different RBF center widths were assigned

for the two approaches. In [113], the input-output clustering outperformed the input clustering in function prediction on two datasets. Besides Publication VII, we are not aware of any comparisons on classification tasks.

5.2.2.3 Class-specific Clustering

In Publication VII, we introduced supervision into RBFNN training simply by clustering items class-by-class. We call the resulting approach class-specific clustering. Our motivation was the success of several *divide-and-conquer* methods, which obtain a better final solution by dividing a complicated problem into several simpler subproblems. In this case, it is significantly easier to perform clustering in a single class only. Even though the clustering must be performed separately for each class, the smaller number of items to be considered in each clustering guarantees a clear saving in the overall computational requirement. The class-specific clustering approach has been applied in some earlier studies [34, 50–52]. However, no comparisons against the traditional clustering approaches are carried out, and the number of applications exploiting the class-specific clustering is still marginal.

After the initial clustering, the results of the class-specific clustering can be further improved by considering the centroid distribution. If two centroids for different classes happen to be co-located or very close to each other, it clearly indicates that the surrounding area should be modeled with more RBF centers in order to better discriminate these classes. A possible rule for handling overlapping clusters from different classes is presented in [34]. However, in Publication VII, we did not apply any post-processing, as we wanted to clearly see the impact of the plain clustering approaches.

5.2.2.4 On Time and Memory Complexity

The time and memory complexities of most clustering algorithms depend on the number of items I, the number of clusters K, and the data dimensionality d (e.g., $O(IKd)$ for a linear algorithm). The data dimensionality, d, for the input and class-specific clustering approaches is directly the dimensionality of the input feature vectors, whereas for the input-output clustering approach the data dimensionality is increased when creating its input vectors according to Eq. (5.10). The final dimensionality is the sum of the input vector dimensionality and the number of classes. Class-specific clustering must be repeated for all C classes, but the number of items and clusters per clustering is smaller. If it is assumed that items are uniformly distributed between classes, the total number of clusters is the same as for input clustering, and an equal number of clusters is assigned for each class, $I^{CS} = I^I/C$ and $K^{CS} = K^I/C$, where CS and I refer to the class-specific and input clustering approaches, respectively.

The final cost of each clustering approach is determined by the selected clustering algorithm and the data properties. For most real-life benchmark datasets, $d^{IO} \approx d^I$ (IO refers to the input-output clustering) and, thus, the complexities of input and input-output clustering approaches are about the same. However, for some datasets, $d^{IO} > 2d^I$. In the latter situation, the input-output clustering would take more than twice the time required for the input clustering approach even when a clustering algorithm with a linear complexity is applied. The class-specific approach is C times faster than the input-output clustering with an algorithm having a linear complexity, if the clustering algorithm is not otherwise affected by the clustering approach. More benefit is gained if a slower algorithm

is used. Furthermore, the class-specific approach may reduce the number of required iterations, which will further increase the computational benefit.

5.2.2.5 Experimental Comparison Results

In Publication VII, we performed an extensive comparison of the input, input-output, and class-specific clustering approaches over 25 benchmark classification datasets. We applied three different clustering algorithms, namely K-means, APC_III [52], and MD-PSO. For K-means, three different numbers of clusters were used: $3C$, $\max(5C, \sqrt{I})$, and $I/3$. For MD-PSO clustering, the winner of the fitness function comparison of Publication VIII discussed in Section 4.3, the Xu index (Eq. (4.4)), was used as the fitness function. However, we did not use the FECC approach, as it was not published at the time. Eq. (4.3) was used as the fractional fitness function. The results were evaluated by the classification accuracy and the geometric mean of recall values on the test set.

APC_III method and K-means with $I/3$ clusters produced the highest numbers of RBF neurons. There was no statistically significant difference between the different clustering approaches for these highest neuron numbers. Therefore, as the class-specific approach is computationally the most efficient, it should be applied. For smaller numbers of neurons, the class-specific approach also resulted in better classification results.

For MD-PSO and K-means with $3C$ and $\max(5C, \sqrt{I})$ clusters, we detected statistically significant difference in the overall results. Therefore, we further analyzed the pair-wise differences using Wilcoxon Signed-Ranks test. For 25 datasets, the null hypothesis can be rejected at the 0.05 significance level if T is less than 89 and at the 0.01 significance level if T is less than 68. The obtained T-values are given in Table 5.3. On each line, we compare two clustering approaches for one clustering method. Classification accuracy and geometric mean of recall values are considered separately. The better performing approach is bolded and the corresponding T-value is bolded if it signals that the null hypothesis can be rejected at the 0.05 significance level. When the input and input-output clustering approaches are compared with MD-PSO and K-means with $3C$ clusters, the winner approach is different for the classification accuracy and the geometric mean values. Therefore, the table has two lines for these comparisons. However, the differences in these cases are insignificant.

Clearly, in addition to its easier computation, the class-specific clustering approach resulted in significantly better clustering results when the final number of neurons was relatively small. Between the input and input-output clustering approaches, there were no significant differences. However, we observed that the input-output clustering approach can be beneficial when the number of classes is low in comparison with the feature vector dimension. When the number of classes is high, the input-output clustering approach can make the results deteriorate.

Table 5.3: Wilcoxon Signed-Ranks test T-values for comparisons performed with different clustering algorithms and performance measures (Acc.=accuracy, G-mean=geometric mean of recall values, CS=class-specific clustering, I=input clustering, IO=input-output clustering) [Publication VII]

Approach 1	**T (Acc.)**	**T (G-mean)**	**Approach 2**
		MD-PSO	
CS	**34**	**46.5**	IO
CS	**57**	**68.5**	I
IO	141		**I**
IO		154.5	I
		K-means with 3C clusters	
CS	**25**	**17.5**	IO
CS	**27**	**9.5**	I
IO	143		I
IO		146.5	**I**
		K-means with $\max(\mathbf{5C}, \sqrt{\mathbf{I}})$ **clusters**	
CS	**56**	**55.5**	IO
CS	**64**	**49.5**	I
IO	129	144.5	**I**

6 Collective Network of Binary Classifiers

CNBC has strong similarities to the ensembles of classifiers. The main idea of the ensembles of classifiers comes from the observation that different classifiers typically make mistakes on different samples. If the error rates of individual classifiers are below 50%, it is likely that the majority vote of the ensembles will result in the correct classification decision. Also in practice, the ensembles of classifiers typically perform much better than the individual classifiers that make them up [27].

To form a successful ensemble, classifiers should be accurate enough and diverse. The diversity can be obtained in several ways. The classifiers can be of different type, but also a single classifier type can provide diversity if the classifiers are based on different feature sets or if they are trained using different subsets of the training samples [27]. Another important question is how to make the final classification decision. The most typical approach is majority voting, but there are also several other options, which can be used if the classifiers output also the probabilities of each class [72]. One possibility is to give the outputs of the initial classifiers as inputs to a separately trained output classifier [49].

The term ensemble of classifiers typically refers to classification systems where all the individual classifiers simultaneously classify all the considered classes. It is also common to divide the classification task into simpler binary classification problems. Each classifier concentrates on a *one-versus-all* or *one-versus-one* classification subtask and the final decision is reached considering all the binary decisions. A well-known example is the multiclass Support Vector Machine (SVM) [46]. Typical aggregation techniques include binary voting, i.e., MaxWins rule, for one-against-one classification and maximum confidence strategy for one-against-all classification, but several other techniques have been suggested. [39]

For any classification system, possibility for incremental training is a desired property. This means that, when new data are added to the training set, it is not necessary to retrain everything, but the new information can be learned on the top of the previously obtained learning. The new data may mean new samples from the current classes, new samples from a currently unknown class, or new features. New samples from the current classes are easiest to accommodate and techniques are abundant. Also the addition of new classes has been considered in several works. In [97], Polikar et al. introduced a type of ensemble of classifiers, Learn++, that can be trained incrementally and also accommodate later added new classes. The main idea of Learn++ is to select the training subset used to train each new classifier so that samples misclassified by the ensemble of the previously trained classifiers are more likely to be included. The class-incremental training of one-against-all multiclass SVM is proposed in [127]. To the best of our knowledge,

CNBC was a pioneer work in the sense that, in addition to new samples and new classes, it can also incorporate new features without losing the already acquired learning.

CNBC is a semantic dynamic classification system based on the *divide-and-conquer* paradigm. Learning of different classes is distributed to binary one-against-all classifiers and, if there are several features available, the learning based on each feature is further distributed for separate binary classifiers. Thanks to this approach and the applied evolution technique, CNBC can incrementally learn both new classes and new features. In Publication II, the CNBC classifier was used in the general image classification and retrieval (see Section 6.2). In [70], the focus was especially on incremental training results. In addition to general image databases, CNBC has been applied to databases of macroinvertebrate images [61], polarimetric Synthetic Aperture Radar (SAR) images [65, 111], and audio [68].

6.1 CNBC Topology

CNBC encapsulates several Networks of Binary Classifiers (NBCs) and each NBC corresponds to a different semantic class, $c \in \{1, \ldots, C\}$, where C denotes the total number of classes. They, in turn, contain a number of evolutionary Binary Classifiers (BCs) corresponding to available features, $f \in \{1, \ldots, F\}$, where F denotes the number of features. A single BC, BC_f, in NBC_c is thus trained to solve the binary (i.e., two-class) classification problem of whether its inputs belong to class c according to feature f. The outputs of all the BCs within NBC_c are evaluated with a specific fuser BC. The inputs of the fuser BC are the outputs of the other BCs and it has a single binary output indicating the relevance of each media item to class c. The main idea of the CNBC framework is to avoid the need of very complex classifiers by distributing the massive learning task between numerous simple (or at least significantly simpler) BCs. The overall CNBC topology is illustrated in Fig. 6.1.

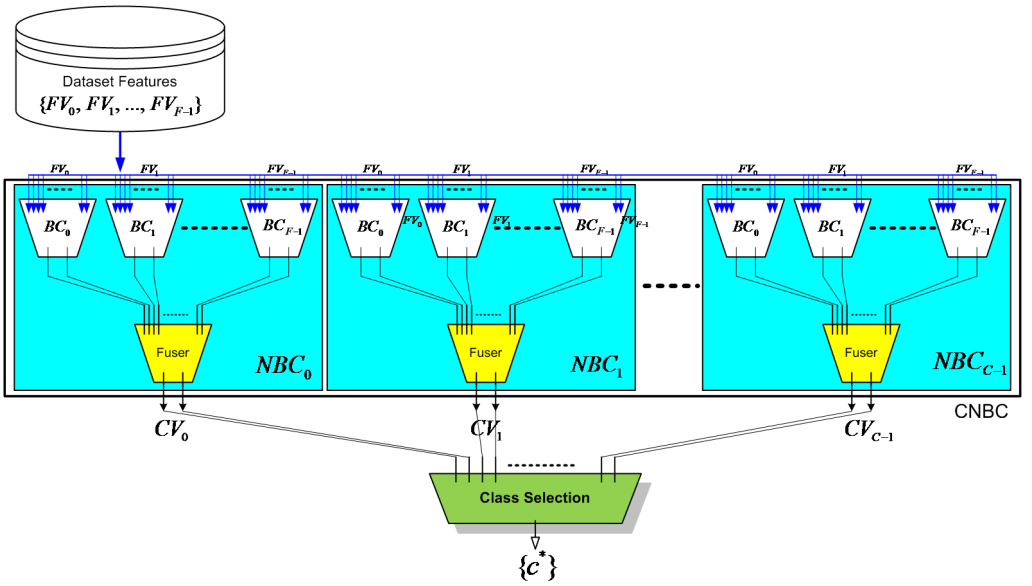


Figure 6.1: Topology of the CNBC framework with C classes and F features [Publication II]

Due to the scalability of CNBC it is easy to add or remove classes, i.e., NBCs, if the database classes are modified. Furthermore, if new features are extracted, new BCs can be similarly added in each NBC in order to take advantage of the new feature. When such a modification is done, the existing NBCs or BCs need to be evolved only if their performance is affected. Any classifier type and/or training method can be used for the BCs. Using MLPs trained with MD-PSO as explained in Section 5.1 enables efficient incremental training of the classifier. In addition to the current best network topologies and parameters, the best networks in other solution space dimensionalities (i.e., network topologies) are saved. When an existing BC needs to be further evolved after a change in classes or features, a MD-PSO swarm is initialized randomly as usual except for a single particle, which is placed to the previous best position (in all the considered dimensionalities). This allows the training procedure to widely exploit information gathered during the previous training iterations, while it can still effectively adapt to the updated solution space.

The evolution of the CNBC classifier is performed separately for each NBC following a two-phase training strategy. In the first phase, all the BCs in the input layer are trained. In the second phase, the training samples are propagated through the best BC configurations found in the first phase and the outputs are concatenated to form the input features for evolving the fuser BCs. In this way, the fuser BCs learn the significance of each individual BC (or the corresponding feature) for the discrimination of that particular class. Some features may not be very discriminative or useful at all for certain classes and, during the second training phase, the fuser BCs can learn which features are the most relevant for each class.

6.2 Experimental Results

In Publication II, CNBC was used for classification over two image databases: *Corel_10* consisting of 10 classes and 1000 images and *Corel_Caltech_30* consisting of 30 classes and 4245 images. On both databases, we evolved CNBC using MD-PSO and exhaustive BP applied separately over each network architecture. Also two different feature sets with 7 (188 elements in total) and 14 (2335 elements) features were used. The results are summarized in terms of MSE and Classification Error (CE) in Table 6.1.

The results indicate that CNBC does not suffer from the major dimension increase (from 188-D to 2335-D). Instead, it shows better generalization ability with the larger feature set. Thanks to the *divide-and-conquer* paradigm, CNBC can benefit from each new feature. On *Corel_10*, MD-PSO training gives better training results, while BP outperforms MD-PSO on the test set. Most likely overfitting to the training data occurs for MD-PSO. On the larger *Corel_Caltech_30* database, exhaustive BP yields better training results than MD-PSO, but the test results are similar. The test classification error for both training methods has increased in comparison to the results on *Corel_10*. This is an expected result for a larger database.

For *Corel_10*, we also experimented with incremental training and performed testing in 3 separate stages: classes 1-5, classes 6-8, and classes 9-10. The existing NBCs were further evolved only if their performance fell below the minimum classification accuracy threshold. The results labeled as *Corel_10* incr. in Table 6.1 are the final results after all three stages. The statistics have slightly deteriorated, which is expected as some retraining of existing NBCs was purposefully skipped.

Table 6.1: Classification statistics for CNBC [Publication II]

Database	No. feat.	Method	Train MSE	Train CE	Test CE
Corel_10	7	MD-PSO	**0.32**	**3.55**	20.18
		BP	0.45	4.88	**18.36**
	14	MD-PSO	**0.28**	**3.18**	18.63
		BP	0.34	3.33	**14.54**
Corel_10 incr.	7	MD-PSO	1.36	6.89	28.63
		BP	**0.82**	**4.22**	**21.63**
	14	MD-PSO	1.23	**6.66**	26.36
		BP	**0.91**	7.55	**23.81**
Corel_Caltech_30	7	MD-PSO	0.54	8.10	**33.40**
		BP	**0.24**	**2.95**	34.67
	14	MD-PSO	0.33	5.47	36.33
		BP	**0.074**	**1.31**	**33.86**

Table 6.2: CBIR statistics for CNBC [Publication II]

No. features	Evol. method	*Corel_10*		*Corel_Caltech_30*	
		ANMRR	AP	ANMRR	AP
7	MD-PSO	33.09	64.01	43.04	54.47
	None	55.81	42.15	60.21	37.80
14	BP	22.21	76.20	32.00	65.37
	None	47.19	50.38	62.94	34.92

In Publication II, CNBC was also applied for Content-based Image Retrieval (CBIR) by using its outputs as new features. The results are given in terms of Average Normalized Modified Retrieval Rank (ANMRR) and Average Precision (AP) in Table 6.2 accompanied with results obtained by using the original low level features (Evol. method: None). It is evident from the results that CNBC has significantly enhanced the retrieval results regardless of the evolution method.

7 Feature Synthesis

CBIR and image classification systems generally operate on image features and their performance is limited by the features' discrimination power. Especially low-level features automatically extracted from images are usually not discriminative enough to fully enable distinction among images belonging to different classes. Visually very similar images may belong to different classes or, vice versa, images belonging to the same class may have completely different features. Higher level of understanding of image content is required to understand the correct classes. This is known as the *semantic gap* problem. To overcome the semantic gap, ground truth information is gathered from users. Several different techniques to exploit the gathered information have been proposed. For example, feature selection uses image labels to select features best discriminating certain classes from each other. The original features are not changed, but the purpose is to find a subset of features that best fits the current application and database.

Feature synthesis is a more advanced tool to tackle the semantic gap. It aims at transforming the original features into more discriminative ones by applying some arithmetic operations. This is illustrated in Fig. 7.1, where a successful feature synthesis operation is applied on 2D features of a 3-class database. The discrimination power of the features has clearly increased. In the figure, the dimensionality of the new synthesized features equals to the original dimensionality, but the new features can also have a different dimensionality. Even a very limited set of operators brings about an enormous number of possible ways to combine elements of the old feature vectors. Together all the options form a complicated multimodal search space. Therefore, the first efforts to carry out feature synthesis were based on different EAs [7].

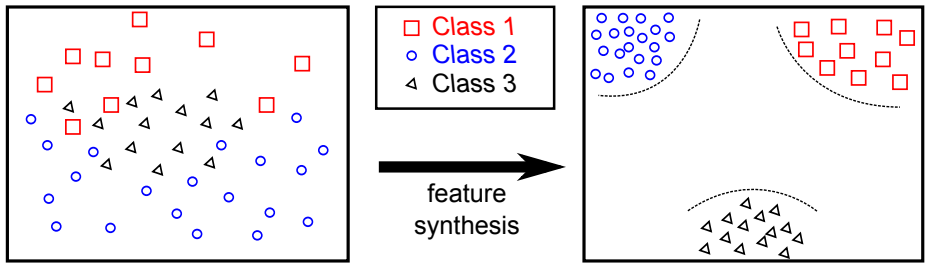


Figure 7.1: An illustrative feature synthesis, which is applied to 2D feature vectors of a 3-class database [Publication VI]

7.1 Related Work

Already in 1983, Michalski proposed a methodology for inductive learning, which can be considered as feature synthesis approach [85]. In his work, inference rules were used to learn new structural symbolic descriptions from original observations. Sherrah et. al [105, 106] used GP to combine selected original measurements with selected operators. In their work, individuals were parse trees for functional expressions. The leaf nodes were original measurements, constants, and statistics. The internal nodes were different functions operating on the leaf node inputs. The features formed by the individuals were evaluated according to the classification error obtained using simple classifiers. The method was computationally heavy, but the classification error was significantly improved in comparison to the original measurements classified with the same classifiers. In [95], a similar GP-based method was applied on audio signals to automatically evolve music descriptors. A feature generation method based on GA was introduced in [99] and applied on artificial data and chromatography. The term *feature synthesis* was not used in these early works, but different terminology was used to refer to the similar concepts, e.g., inductive learning, automatic feature selection and generation, or automatic evolution of descriptors.

The term feature synthesis was adopted in [10, 125], where the first efforts in feature synthesis for images were presented. GP was utilized to synthesize features for face expression recognition. The individuals were composite operators represented by binary trees whose internal nodes were primitive operators and leaf nodes were primitive features. The primitive features were generated by filtering the original images using a Gabor filter bank at 4 scales and 6 orientations (i.e., 24 images per an original image) and the primitive operators were selected among 37 different options. To evaluate the fitness of individuals, a Bayesian classifier was trained simultaneously with evolving the feature synthesis and its classification accuracy in the training set gave the fitness values. Finally, the whole database was synthesized using the best individual, and the corresponding Bayesian classifier was used to classify the images into 7 expressions classes. In comparison to similar classification methods not using feature synthesis, this approach brought only a minor improvement to the expression recognition rate.

In [80], a similar feature synthesis technique based on Co-evolutionary Genetic Programming (CGP) was applied on object recognition in SAR images. Now separate subpopulations were used to evolve several feature synthesis operations, but the primitive features were only one-dimensional properties computed from the images, and thus, each operation produced a single one-dimensional composite feature. At the end, 20 feature synthesis operations produced by 20 subpopulations were applied on all images and the features were combined into 20-dimensional feature vectors. The classification accuracy obtained with the synthesized features only occasionally surpassed the accuracy obtained directly with the primitive features. The same technique was applied in [30] for image classification and retrieval. The CGP was used to evolve 10-dimensional feature vectors from the original 40-dimensional feature vectors. The classification accuracy obtained using the new features was compared against the performance of a classifier using 10-dimensional feature vectors generated by Multiple Discriminant Analysis (MDA) and a SVM classifier using the original 40-dimensional feature vectors. The features synthesized by CGP turned out to perform better than the features produced by MDA in all the tests. Compared to the SVM classifier, the results were similar or better when the database classes consisted of multiple clusters in the original feature space.

In [76, 77], the CGP-based features synthesis method and the Expectation-Maximization (EM) algorithm were combined into Co-evolutionary Feature Synthesized Expectation-Maximization (CFS-EM). CFS-EM first uses a minor part of the training data to reduce the feature space dimensionality and simultaneously learns an initial Bayesian classifier using the CGP-based feature synthesis method. Then the whole database (the rest may be unlabeled) is synthesized into a lower dimensionality, and finally, the classifier is refined with the EM algorithm. The classification and retrieval results obtained by CFS-EM were both improved compared to the CGP-only approach.

While the number of methods specially developed for feature synthesis is quite modest, several classifier types, such as ANNs and SVMs, can also be considered as feature synthesizers. For example, ANNs take the original features as inputs and try to learn such internal parameters that would transform the inputs into 1-of-C-encoded (where C is the number of classes) output vectors. In the optimal case, the network defines a synthesizer which can transform all inputs into the corresponding class vector (i.e., $[1, 0, ..., 0]$ for the first class and so on), and thus, the discrimination power of the features has increased significantly. The simplest ANNs, SLPs, form in each neuron a weighted sum of all input vector elements and pass it through a bounded non-linear function (e.g., hyperbolic tangent or sigmoid) to get one of the output vector elements. When the network is trained, only the network weights and biases are optimized. Otherwise, the synthesis based on SLPs follows a fixed path. MLPs have a more complicated network structure, but they similarly learn a feature synthesis operation via weight optimization.

7.2 Evolutionary Feature Synthesis Using MD-PSO

In Publication III and Publication V, we proposed using MD-PSO for Evolutionary Feature Synthesis (EFS). Our objective was to define an EFS method that can

- perform an optimal feature selection,
- search for optimal weights for each selected feature,
- search for optimal arithmetic, linear or non-linear, operators,
- search for the optimal output feature vector dimensionality,
- using any given fitness function to measure the quality of the solution.

To achieve the objectives, we used MD-PSO to

1. select $W + 1$ original (or already synthesized) feature elements, $f_0, ..., f_W$,
2. scale the selected features using proper weights, $w_0, ..., w_W$,
3. select W operators, $\Theta_1, ..., \Theta_W$, to be performed over the selected and scaled features,
4. bound the results using a non-linear operator (e.g., hyperbolic tangent, tanh).

If the application of a specific operator, Θ_i, on features f_j and f_k is denoted as $\Theta_i(f_j, f_k)$, a formula for the overall synthesis of a single element of the output feature vector can be given as

$$y_j = \tanh(\Theta_W(...\Theta_2(\Theta_1(w_0 f_0, w_1 f_1), w_2 f_2), ...), w_W f_W). \quad (7.1)$$

In other words, the operator Θ_1 is first applied to the scaled features f_0 and f_1, then operator Θ_2 is applied to the result of the first operation and the scaled feature f_2 and so on, until the last operator Θ_W is applied to the result of the previous operations and the scaled feature f_W. The positional PSO process within MD-PSO can select the proper feature elements, weights, and operators, while the dimensional PSO process can optimize the dimensionality of the synthesized feature vector (i.e., the number of feature elements). The MD-PSO-based EFS can also be applied iteratively. The output of the previous synthesis is simply given as the input for the next EFS run. The number of runs, R, can be either specified in advance or adaptively determined so that more runs are carried out until the fitness improvement is no longer significant.

The above described feature synthesis can be seen as a generalization of ANNs. A SLP performs steps 2 and 4, but it does not apply feature or operator selection. A SLP also adds a bias value to the sum of scaled features. To allow a similar mechanism, we complement each input feature vector by a constant value of one. When a bias is beneficial, MD-PSO can select this constant value instead of selecting one of the actual feature vector elements, scale it, and combine it with other scaled features. Instead of summing the bias as in SLPs, it may be combined using also another operator. Now setting R to 1, $W+1$ to be the dimensionality of input features, d, each feature selection as $f_i = f_{[i]}$, where $f_{[i]}$ denotes the i^{th} element of the input feature vector, and each operator Θ_j to be "+" limits the MD-PSO-based EFS technique to be equivalent to a SLP. Performing several EFS runs leads to a system similar to a MLP. The output dimensionality of ANNs is fixed, but MD-PSO can optimize also the output feature dimensionality. Furthermore, a major difference between MLPs and MD-PSO-based EFS is that MD-PSO evaluates the fitness of the synthesized feature vectors at every run, while in a MLP only the final fitness score (i.e., MSE between the actual and target output vectors) is considered.

7.2.1 MD-PSO Particle Encoding in EFS

The MD-PSO particles should be encoded in such a way that each particle position, $\mathbf{x}_p^d$, represents a full set of parameters required for feature synthesis. The search space dimensionality corresponds to the dimensionality of synthesized features. A position in that dimensionality should then give all the information required to carry out feature selection, feature weighting, and operator selection to form all the output elements. Accordingly, each element of a position, $\mathbf{x}_{p,j}^d$, defines a way to synthesize the j^{th} output feature element. It should encapsulate the $W+1$ features, $W+1$ weights, and W operators needed to perform the described synthesis. We encode it as a $2W+1$ dimensional vector with $W+1$ A-type and W B-type elements, which define the corresponding feature synthesizer parameters as follows,

$$\begin{aligned} f_i &= \lfloor A_i \rfloor + 1, i \in 0, ..., W \\ w_i &= A_i - \lfloor A_i \rfloor, i \in 0, ..., W \\ \Theta_j &= \lceil B_j \rceil, j \in 1, ..., W, \end{aligned} \tag{7.2}$$

where $\lfloor x \rfloor$ and $\lceil x \rceil$ denote the *floor* and *ceiling* operators, respectively. The ranges for the A- and B-type elements are set according to the dimensionality of the input feature vector, d, and the number of operators available, U, i.e., $A_i \in [0, d[$ and $B_j \in]0, U]$. The weights, w_i, are thus limited to the range $0 \leq w_i < 1$.

The particle encoding is illustrated in Fig. 7.2, where a position of particle p in dimensionality 6, $\mathbf{x}_p^6$, is shown. As the position dimensionality is 6, the synthesized feature vector,

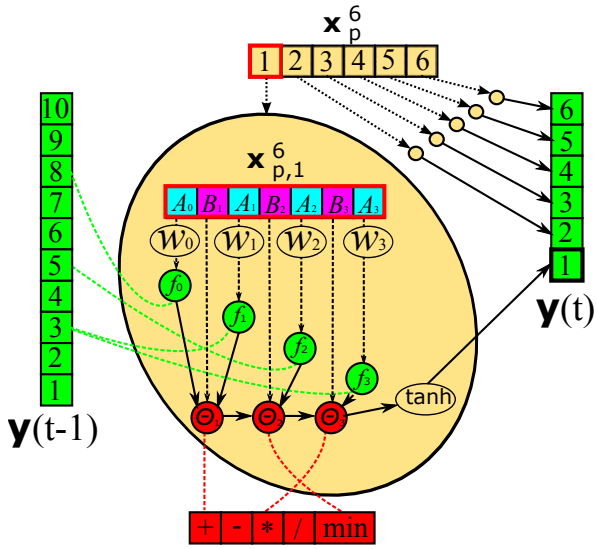


Figure 7.2: An illustrative example of particle encoding of a position in dimensionality 6. W is set to 3.

$\mathbf{y}(t)$, will have six elements and each feature vector element will be synthesized according to the corresponding position element. In the figure, W is set to 3, which means that 3+1=4 elements of the input feature vector, $\mathbf{y}(t-1)$, will be combined using 3 operators, and the dimensionality of each element of the position, $\mathbf{x}^6_{p,j}$, is 7 ($2W+1=7$). Only the first element, $\mathbf{x}^6_{p,1}$, is fully shown. If the i^{th} element of the input feature vector is denoted as $f_{[i]}$, the formula for the synthesis of the first output feature defined by $\mathbf{x}^6_{p,1}$ can be given as

$$y_j = \tanh(\min((w_0 f_{[8]} + w_1 f_{[3]}), w_2 f_{[5]}) * w_3 f_{[3]}). \tag{7.3}$$

7.2.2 MD-PSO Fitness Evaluation in EFS

In Publication III, the synthesized features were used in image retrieval. There, AP, or more precisely its additive inverse, was directly used as the fitness function for MD-PSO. It is evident, however, that this fitness function is not applicable on larger databases, because the computation of AP requires querying with every database image. When AP is used for fitness evaluation, this must be done for every particle at every iteration.

In Publication V, we adapted a method similar to the one used in ANNs: We assigned target vectors for each class and then evaluated the fitness in terms of MSE between the synthesized output vectors and the target output vectors. However, we wanted to allow MD-PSO to find the optimal output feature dimensionality and not to decide it *a priori* as in ANNs. Therefore, we generated target output vectors for all dimensionalities within the range $\{d_{\min}, ..., d_{\max}\}$. When generating the target vectors, we tried to achieve the two criteria for a good Error Correcting Output Code (ECOC) suggested in [28], i.e., large row and column separation.

- *Row separation*: Each target vector should be well-separated in the sense of Hamming distance from each of the other target vectors.

- *Column separation*: Each column in the vector table should be well-separated in the sense of Hamming distance from each of the other columns.

With large row separation, the final synthesized vectors can somewhat differ from the target output vectors and still be able to discriminate between classes. Each column of the target vector table can be seen as a different binary classification task. The original classes having value 1 in the specific column form the first metaclass and the rest of the classes having value -1 in that column form the second one. Depending on the similarity of the original classes, some binary classification tasks are likely to be notably easier than others. Since the same target output vectors should be used with any given input classes, and thus, nothing can be assumed about their similarity, it is beneficial to keep the binary classification tasks as different as possible, i.e., maximize the column separation.

There is no simple and fast method available to generate target vectors with maximal row and column separation, but we used the following simple approach to create target output vectors with row and column separations satisfactory for our purposes:

1. Form an empty target vector table with C rows and $d_{\max}$ columns.
2. Assign $b_{\min}$ as the minimum number of bits needed to represent C values. Form a bit table having C rows, $b_{\min}$ columns, and the binary representation of the row numbers $\{0, ..., C-1\}$ as its rows.
3. For each row of the target vector table, assign the first empty $b_{\min}$ values equal to the corresponding row in the bit table.
4. Move the first row of the bit table to the end of the table and shift the other rows up by one row.
5. Repeat the previous two steps until the target vector table is filled.

Experimentally we observed that it is beneficial to add the following steps to the target vector creation procedure:

6. Replace the first C values in each target vector by a 1-of-C-encoded section.
7. Replace each 0 by -1 in the target vectors.

While step 6 reduces row and column separation, it is generally easiest to find a binary classifier that separates a single class from the others. Therefore, adding the 1-of-C-encoded section usually improves the results. The target vectors for dimensionalities below $d_{\max}$ can be obtained by leaving out the excess elements at the end of the target vectors. Since the common elements in the target vectors for different dimensionalities are identical, FGBF can freely combine elements taken from particle positions in any dimensionality.

The target output vector generation for a 4-class case is illustrated in Fig. 7.3. At the beginning, there is a 1-of-C-encoded section of four elements, while the remaining elements are consecutive binary representations of the row numbers 0-3 (i.e., $b_{\min} = 2$). $d_{\max}$ is set to 10. For the sake of clarity, the elements set to -1 are shown as empty boxes.

t_1	1				1		1	1		
t_2		1			1	1				1
t_3			1					1	1	
t_4				1		1	1		1	1

Figure 7.3: A sample target vector encoding for 4 classes.

Using the created target vectors, the fractional fitness scores for FGBF are computed as

$$f_j\left(\mathbf{x}_{p,j}^{d_p}(t)\right) = \sum_{k=1}^{C} \sum_{\forall \mathbf{y} \in c_k} \left(t_{k,j} - y_j\right)^2, \tag{7.4}$$

where $t_{k,j}$ and y_j denote the j^{th} elements of the target output vector for class c_k and the synthesized output vector, respectively. To compute the overall fitness score, the fractional fitness scores for different dimensionalities are added together and the score is then normalized with respect to the number of dimensionalities. As mentioned earlier, we observed that the first C elements are usually the easiest to encode due to the 1-of-C-encoded target vector section. To slightly increase the probability of MD-PSO to converge on higher dimensions, we handle the first C elements separately in the summation and favor the remaining dimensionalities, $d > C$, by strengthening their normalizing divisor with an additional power parameter, $\alpha > 1$. We also set $d_{\min} > C$. Consequently, the overall fitness function for MD-PSO can be given as

$$f\left(\mathbf{x}_p^{d_p}(t)\right) = \frac{1}{C} \sum_{j=1}^{C} \sum_{k=1}^{C} \sum_{\forall y \in c_k} \left(t_{k,j} - y_j\right)^2 + \frac{1}{(d_p - C)^{\alpha}} \sum_{j=C+1}^{d_p} \sum_{k=1}^{C} \sum_{\forall y \in c_k} \left(t_{k,j} - y_j\right)^2. \tag{7.5}$$

7.2.3 Experimental Results

The MD-PSO-based EFS where 1-AP was used as the fitness function was tested in Publication III on a 5-class database with 100 images in each class. 45% of the database was used for evolving the EFS. Retrieval results using the original (RGB histogram) and synthesized (after one and three EFS runs) features in terms of the standard ANMRR measure are given in Table 7.1. The discrimination power of the features clearly increased after applying EFS, but as mentioned, the computational effort required for all the fitness evaluations is so large that the method is not feasible for much larger databases.

In Publication V, the MD-PSO-based EFS was used with the improved fitness evaluation. CBIR results on a 10-class database with 100 images in each class are given in Table 7.2 in terms of ANMRR and classification results in Table 7.3 in terms of test classification

Table 7.1: CBIR results on a 5-class database [Publication III]

Features	**ANMRR**
Original RGB	0.472
Synt. 1^{st} run	0.392
Synt. 3^{rd} run	0.277

Table 7.2: CBIR results (ANMRR) on a 10-class database [Publication V]

Features	RGB	YUV	LBP	Gabor	all
Original	0.589	0.577	0.635	0.561	0.504
Synt. 1^{st} run	0.500	0.505	0.519	0.520	0.357
Synt. best run	0.385	0.397	0.396	0.428	0.280

Table 7.3: Test classification error on a 10-class database [Publication V]

Features	RGB	YUV	LBP	Gabor
Original	0.420	0.371	0.560	0.433
Synt. 1^{st} run	0.312	0.307	0.422	0.367

error. Again 45% of the database was used for evolving the EFS. In CBIR experiments, the whole database was considered, while classification was conducted in the remaining 55% only.

Both CBIR and classification results exhibit a clear increase in discrimination ability. It was also observed that while the CBIR results kept improving during several EFS runs, the classification results barely improved after the first run. Most likely, the EFS overlearns the training data on later runs. It can help further improve the CBIR results as the training set is also considered, but the test set features are not improving anymore. On larger databases, overlearning might not be encountered so early. However, initial testing on much larger databases showed that MD-PSO-based EFS was no longer performing well. Possible reasons are considered in the next section.

7.3 Feature Synthesis via One-against-all Perceptrons

The MD-PSO-based feature synthesis performs well on small (e.g., 1000 images) databases, but the results are not equally good on larger databases. We assume that the performance deteriorates because the method attempts to find a synthesizer which can simultaneously discriminate all the classes from each other. When the number of classes grows, finding a good solution capable of doing that soon becomes a practically impossible problem for any optimization method. Also the GP-based methods introduced in Section 7.1 follow a similar approach, where all the classes are considered simultaneously no matter how many and how complex classes there may be. Their application is further limited by their computational complexity, which makes it infeasible to significantly increase database size. In Publication VI, we proposed a simple alternative, which relies on the *divide-and-conquer* paradigm and is significantly faster than the earlier feature synthesis techniques.

The feature synthesis technique proposed in Publication VI transforms the original features using parallel one-against-all perceptrons. We call it Perceptron Feature Synthesis (PFS). Each perceptron is trained to discriminate a single class from the rest. During the training phase, only the images belonging to that particular class are considered positive samples and all the other images negative samples. For perceptrons, we use the typical 1-of-C-encoded target output (i.e., [1 -1] for positive samples and [-1 1] for negative samples). After the training phase, the new synthesized features can be formed by propagating the

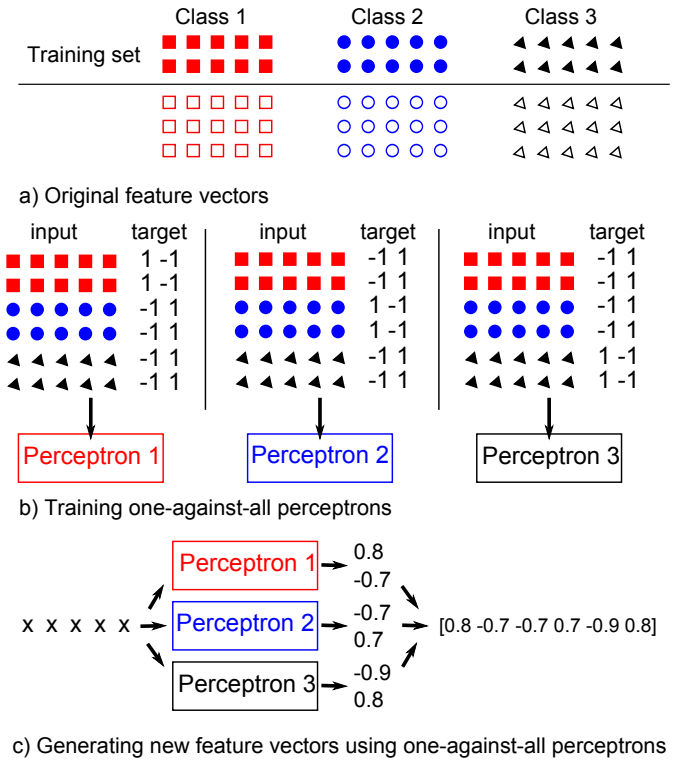


Figure 7.4: Feature synthesis via parallel one-against-all perceptrons for a simplified 3-class database with 5D original feature vectors [Publication VI]

original features through all the perceptrons and concatenating the outputs. In that way, the dimensionality of the synthesized features is always twice the number of classes.

The whole PFS process is illustrated in Fig. 7.4. In the figure, the database consists of three classes with five items in each class. However, the class information is only needed for the training set and PFS can be applied also when the majority of the database is unlabeled. The original feature vector dimensionality is five. For three classes, three one-against-all perceptrons are trained. For the first perceptron, only the training items from class 1 are given as positive samples (their target output vector set to [1 -1]). Similarly, for the second perceptron only items from class 2 are considered positive samples and for the third perceptron only items from class 3. Finally, after training all the perceptrons, all original feature vectors (belonging to both train and test sets) are propagated through all the three perceptrons and the outputs are concatenated to form new 6-dimensional feature vectors. As in the figure, the perceptron outputs are typically not 1 and -1 but somewhere between these two extremes. This may be also a desired property as it preserves some intra-class variations. For example in CBIR, a user probably prefers receiving the most similar database items within a class first.

It is also possible to incrementally train the PFS system. When a new class is added to the database, a new perceptron will be trained for the new class. The previously trained perceptrons need to be retrained only if they fail to classify the new samples with a required accuracy. Training of the affected perceptrons may also be started from the previous state or even the whole architecture space can be evolved in the same manner as

in the incremental training of CNBC, discussed in Chapter 6. Furthermore, PFS can be performed in several runs by using the synthesized feature vectors from the previous run as the input feature vectors for the next.

7.3.1 Experimental Results

In Publication VI, PFS was applied on low-level features extracted from three general image databases: *C_10* database had 10 classes and 1000 pictures, *CC_30* database had 30 classes and 4245 images, and the largest *F_20* database had 20 classes and 34481 images with a very imbalanced distribution. CBIR results in terms of ANMRR using the original and synthesized features are given in Table 7.4 and test classification errors obtained using the simple Nearest Centroid Classifier (NCC) in Table 7.5. CSD, DOCQ, etc. are different low level features, whose details can be obtained from Publication VI. In column *All*, all the features are used for retrieval/classification with equal weights. In CBIR, the whole databases were considered, while the classification experiments were carried out in the test set only. For *C_10* and *CC_30* databases, 20 PFS runs were applied. For *F_20*, we only performed three runs due to the large database size. In these experiments, MD-PSO was used as the perceptron training method.

The results confirm that PFS can effectively increase the features' discrimination power. As with the MD-PSO-based EFS, it was observed that CBIR results keep improving during several PFS runs, while the classification results mainly improve during the first run. Thus, we assume that the majority of the improvement of the CBIR results happens in the training set, which means that PFS is overlearning it. However, it was also observed that on the larger databases, especially on F_20, overlearning is less of a problem as also the test classification error keeps improving during the later runs.

In Publication VI, we also compared the classification error obtained using a NCC over the synthesized features against some well-known classifiers: MLP, SVM, and Random Forest (RF). The results of these comparisons showed that PFS can improve the discrimination power of the low-level features to such level that after the synthesis even the simple NCC can achieve results comparable to the best results obtained with several state-of-the-art classifiers. Furthermore, the classification results obtained by the NCC and PFS features turned out to vary significantly less among the different low-level features than the results obtained by the other classifiers. As a result, NCC+PFS had the best average classification results. It was also observed that PFS could effectively increase the discrimination power of the features of the minority classes in an imbalanced class distribution.

Finally in Table 7.6, CBIR results obtained using features synthesized by PFS and CNBC outputs as described in Chapter 6 are compared in terms of both ANMRR and AP. The underlying low-level features were the same in both cases. Clearly, the PFS features provide a better performance.

Table 7.4: ANMRR using original and synthesized features [Publication VI]

C_10	**CSD**	**DOCQ**	**EHD**	**HSV**	**LBP**	**ORDC**	**Gabor**	**All**
Original	50.12	73.17	61.68	61.93	68.62	62.28	56.13	43.86
Synt. 1^{st} run	24.73	49.08	47.20	32.47	46.30	42.13	34.61	18.15
Synt. best run	24.73	42.51	41.14	32.40	36.80	34.08	31.24	**17.14**
CC_30	**CSD**	**DOCQ**	**EHD**	**HSV**	**LBP**	**ORDC**	**Gabor**	**All**
Original	67.84	81.84	76.38	78.31	80.67	75.98	70.71	60.39
Synt. 1^{st} run	47.42	72.35	67.92	63.86	67.33	63.88	50.82	41.60
Synt. best run	44.51	63.48	63.49	59.22	56.40	54.51	42.80	**33.49**
F_20	**CSD**	**DOCQ**	**EHD**	**HSV**	**LBP**	**ORDC**	**Gabor**	**All**
Original	71.21	75.74	70.12	74.96	70.89	71.91	72.57	69.91
Synt. 1^{st} run	66.23	72.66	66.17	69.63	65.17	64.42	64.79	61.52
Synt. best run	65.99	72.29	66.06	69.18	64.55	63.40	64.08	**60.21**

Table 7.5: Test classification errors of the nearest neighbor classifier using original and synthesized features [Publication VI]

C_10	**CSD**	**DOCQ**	**EHD**	**HSV**	**LBP**	**ORDC**	**Gabor**	**All**
Original	31.82	50.36	45.09	44.55	62.36	54.36	43.27	20.91
Synt. 1^{st} run	25.29	43.56	38.31	33.04	35.87	32.18	29.67	14.55
Synt. best run	25.29	43.45	38.31	33.04	35.87	31.78	29.67	14.55
CC_30	**CSD**	**DOCQ**	**EHD**	**HSV**	**LBP**	**ORDC**	**Gabor**	**All**
Original	62.39	77.50	77.80	71.00	84.72	77.84	65.28	51.72
Synt. 1^{st} run	46.93	61.05	58.40	60.16	55.40	51.13	41.02	30.69
Synt. best run	46.93	60.53	56.78	57.47	52.48	49.80	40.86	29.99
F_20	**CSD**	**DOCQ**	**EHD**	**HSV**	**LBP**	**ORDC**	**Gabor**	**All**
Original	75.64	97.31	85.65	91.68	77.03	81.30	77.73	73.57
Synt. 1^{st} run	46.22	67.09	87.05	53.66	75.02	87.29	82.32	57.02
Synt. best run	43.90	55.55	86.25	51.02	68.60	44.95	50.28	44.21

Table 7.6: ANMRR and AP using features synthesized by PFS and CNBC outputs [Publication VI]

Method	***C_10***		***CC_30***	
	ANMRR	**AP**	**ANMRR**	**AP**
PFS	**17.14**	**80.86**	**33.49**	**63.17**
CNBC (MD-PSO)	31.09	65.01	43.04	54.47
CNBC (BP)	23.86	74.26	46.44	52.21

8 Conclusions

This thesis shows that Multidimensional Particle Swarm Optimization (MD-PSO) is indeed a versatile tool for different machine learning applications. The thesis covers applications of MD-PSO to dynamic optimization, clustering, training both Multilayer Perceptrons (MLPs) and Radial Basis Function Neural Networks (RBFNNs) for classification, image retrieval and classification using the Collective Network of Binary Classifiers (CNBC) framework, and finally feature synthesis. MD-PSO is a global optimization technique with a special ability to optimize the solution dimensionality simultaneously with optimizing the solution parameters in that dimensionality. This ability is its greatest advantage in comparison to other Particle Swarm Optimization (PSO) variants. It allows to optimize, for example, the number of clusters, the network architecture for MLPs and RBFNNs, and the dimensionality of the features produced by feature synthesis as a natural part of the optimization process.

In most considered applications (except MLP training), MD-PSO was accompanied by Fractional Global Best Formation (FGBF). FGBF is a plug-in to (MD-)PSO that creates at every iteration an artificial global best solution from the best elements of the particle positions. The artificial global best competes against the current global best solution, and if it brings about a higher fitness, it will replace the global best solution in the PSO update functions. FGBF can effectively prevent premature convergence and lead the PSO process faster toward optimal solutions.

Besides highlighting the versatility of MD-PSO and FGBF as tools for machine learning, this thesis also improved the algorithms and the underlying processes in several ways. For dynamic optimization, the MD-PSO algorithm was combined with a multiswarm approach. Multiswarms allow multiple optima to be tracked in dynamic optimization problems. This, in turn, allows the algorithm to rapidly find the new global optimum when the ranking of optima is changed. FGBF, on the other hand, provides a sufficient amount of diversity to maintain the tracking of optima when their locations undergo moderate changes. Compared to earlier PSO multiswarm approaches, FGBF turned out to be the best way to ensure sufficient diversity within multiswarms. Furthermore, the test bench was modified to model also multidimensional dynamic optimization problems, where the dimensionality of the optima can change over time.

A new approach to perform fitness evaluation in Particle Swarm Clustering (PSC), namely Fitness Evaluation with Computational Centroids (FECC), was presented. The FECC approach was experimentally shown to lead to significant improvements in the clustering performance. Any PSC method whose fitness function somehow depends on cluster centroid positions, can be easily made to use FECC.

In RBFNN training, the most important task is to define center locations for Radial Basis Function (RBF) neurons. This is typically done by clustering the input items and placing

RBF centers to the cluster centroids. For this purpose, a novel class-specific approach was proposed. The proposed approach is clearly faster than traditional input and input-output clustering approaches. In the extensive comparisons conducted, it was also shown to significantly improve the classification performance of the RBFNNs especially when the number of Gaussian neurons was kept relatively low.

Finally, two novel feature synthesis techniques were presented. In the first technique, MD-PSO is directly used to search for an optimal synthesis that can transform the original low-level features into more discriminative ones. In the second technique, the feature synthesis is carried out using parallel one-against-all perceptrons. Both techniques were shown to produce a clear increase in the features' discrimination power. The technique based on parallel one-against-all perceptrons is faster and applicable to larger databases.

While the results of applying MD-PSO to various machine learning tasks were satisfactory, the thesis work also showed its limitations. All the datasets considered in this thesis were small in the current research environment, where Big Data has become the default. The presented applications cannot be extended to required data volumes directly. The computational limitations are the most imminent problem, but the global approach of the PSO paradigm is an even more problematic. MD-PSO tries to optimize everything at once. When the number of parameters to be optimized rises to millions, it is simply not possible to handle the resulting search space.

However, I still believe that MD-PSO and similar nature-inspired stochastic optimization algorithms have a future also in the world of Big Data. Already this thesis showed glimpses of what I believe to be the key to their large scale application, e.g., in the class-specific training of RBFNN, in CNBC, and in the feature synthesis based on one-against-all perceptrons. To extend MD-PSO applications toward Big Data, one must *divide* the problem into smaller subproblems and then *conquer* the big problem piece by piece. This will be a major future research topic and will require smaller pieces, novel approaches to find solvable subproblems, and also adaptation of the algorithm itself to allow smoother division and combination of subproblems. This could mean, searching for clever ways to organize the data in smaller chunks, for example, using self-data organization, dividing the training of deep neural networks into smaller tasks where MD-PSO can be exploited, or modifying the algorithm to target cloud computing environments.

Bibliography

[1] A. Abraham, S. Das, and S. Roy. Swarm intelligence algorithms for data clustering. In *Soft Computing for Knowledge Discovery and Data Mining*, pages 279 – 313. Springer US, 2008.

[2] A. Alexandridis and E. Chondrodima. A medical diagnostic tool based on radial basis function classifiers and evolutionary simulated annealing. *Journal of Biomedical Informatics*, 49:61 – 72, 2014.

[3] P. J. Angeline. Tracking extrema in dynamic environments. In *Proc. of International Conference of Evolutionary Programming*, pages 335 – 345, Apr. 1997.

[4] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Perez, and I. Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243 – 256, 2013.

[5] A. Askarzadeh. Comparison of particle swarm optimization and other metaheuristics on electricity demand estimation: A case study of Iran. *Energy*, 72:484 – 491, 2014.

[6] T. Bäck. On the behavior of evolutionary algorithms in dynamic environments. In *Proc. of IEEE International Conference on Evolutionary Computation*, pages 446 – 451, May 1998.

[7] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1 – 23, Mar. 1993.

[8] W. Banzhaf, F. D. Francone, R. E. Keller, and P. Nordin. *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[9] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1 – 127, Jan. 2009.

[10] B. Bhanu, J. Yu, X. Tan, and Y. Lin. Feature synthesis using genetic programming for face expression recognition. In *Proc. of Annual Conference on Genetic and Evolutionary Computation*, volume 3103 of *Lecture Notes in Computer Science*, pages 896 – 907. Springer Berlin Heidelberg, Jun. 2004.

[11] S. A. Billings and G. L. Zheng. Radial basis function network configuration using genetic algorithms. *Neural Networks*, 8(6):877 – 890, 1995.

[12] T. Blackwell. Particle swarm optimization in dynamic environments. In *Evolutionary Computation in Dynamic and Uncertain Environments*, pages 29 – 49. Springer Berlin Heidelberg, 2007.

[13] T. Blackwell and P. Bentley. Dynamic search with charged swarms. In *Proc. of Annual Conference on Genetic and Evolutionary Computation*, pages 19 – 26. Morgan Kaufmann Publishers Inc., Jul. 2002.

[14] T. Blackwell and J. Branke. Multi-swarm optimization in dynamic environments. In *Proc. of EvoWorkshops, Applications of Evolutionary Computation*, volume 3005 of *Lecture Notes in Computer Science*, pages 489 – 500. Springer, Apr. 2004.

[15] T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459 – 472, Aug. 2006.

[16] J. Branke. Moving Peaks Benchmark. http://deap.gel.ulaval.ca/doc/default/api/benchmarks.html#module-deap.benchmarks.movingpeaks. Viewed 6 Oct 2016.

[17] J. Branke. Evolutionary approaches to dynamic optimization problems - a survey. In *Proc. of Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 134 – 137, Jul. 1999.

[18] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proc. of Congress on Evolutionary Computation*, volume 3, pages 1875 – 1882, 1999.

[19] C.-L. Chen, W.-C. Chen, and F.-Y. Chang. Hybrid learning algorithm for Gaussian potential function networks. *IEE Proceedings D - Control Theory and Applications*, 140(6):442 – 448, Nov. 1993.

[20] S. Chen, J. Montgomery, and A. Bolufé-Röhler. Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. *Applied Intelligence*, 42(3):514 – 526, Apr. 2015.

[21] W. N. Chen, J. Zhang, Y. Lin, N. Chen, Z. H. Zhan, H. S. H. Chung, Y. Li, and Y. H. Shi. Particle swarm optimization with an aging leader and challengers. *IEEE Transactions on Evolutionary Computation*, 17(2):241 – 258, Apr. 2013.

[22] C. W. Cleghorn and A. P. Engelbrecht. A generalized theoretical deterministic particle swarm model. *Swarm Intelligence*, 8(1):35 – 59, 2014.

[23] C. W. Cleghorn and A. P. Engelbrecht. Particle swarm convergence: An empirical investigation. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 2524 – 2530, Jul. 2014.

[24] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58 – 73, Feb. 2002.

[25] T. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14(3):326 – 334, 1965.

[26] S. Das, A. Abraham, and A. Konar. Automatic kernel clustering with a multi-elitist particle swarm optimization algorithm. *Pattern Recognition Letters*, 29(5):688 – 699, 2008.

[27] T. G. Dietterich. Ensemble methods in machine learning. In *Proc. of International Workshop on Multiple Classifier Systems*, pages 1 – 15. Springer, 2000.

[28] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2(1):263 – 286, Jan. 1995.

[29] E. Dimitriadou, S. Dolničar, and A. Weingessel. An examination of indexes for determining the number of clusters in binary data sets. *Psychometrika*, 67(1):137 – 159, 2002.

[30] A. Dong, B. Bhanu, and Y. Lin. Evolutionary feature synthesis for image databases. In *Proc. of IEEE Workshop on Applications of Computer Vision*, volume 1, pages 330 – 335, Jan. 2005.

[31] M. Dorigo and C. Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2):243 – 278, 2005.

[32] R. C. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. In *Proc. of Congress on Evolutionary Computation*, volume 1, pages 94 – 100, May 2001.

[33] A. P. Engelbrecht. Particle swarm optimization: Global best or local best? In *BRICS Congress on Computational Intelligence*, pages 124 – 135, Sep. 2013.

[34] M. J. Er, S. Wu, J. Lu, and H. L. Toh. Face recognition with radial basis function (RBF) neural networks. *IEEE Transactions on Neural Networks*, 13(3):697 – 710, May 2002.

[35] P. H. Fidencio, R. J. Poppi, and J. C. de Andrade. Determination of organic matter in soils using radial basis function networks and near infrared spectroscopy. *Analytica Chimica Acta*, 453(1):125 – 134, 2002.

[36] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3 – 14, Jan. 1994.

[37] D. Fouskakis and D. Draper. Comparing stochastic optimization methods for variable selection in binary outcome prediction, with application to health policy. *Journal of the American Statistical Association*, 103(484):1367 – 1381, 2008.

[38] A. Fujita, D. Y. Takahashi, and A. G. Patriota. A non-parametric method to estimate the number of clusters. *Computational Statistics & Data Analysis*, 73:27 – 39, 2014.

[39] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44(8):1761 – 1776, 2011.

[40] V. Gazi. Stochastic stability analysis of the particle dynamics in the pso algorithm. In *Proc. of IEEE International Symposium on Intelligent Control*, pages 708 – 713, Oct. 2012.

[41] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[42] J. Haddadnia, K. Faez, and M. Ahmadi. A fuzzy hybrid learning algorithm for radial basis function neural network with application in human face recognition. *Pattern Recognition*, 36(5):1187–1202, 2003.

[43] H. Haklı and H. Uğuz. A novel particle swarm optimization algorithm with levy flight. *Applied Soft Computing*, 23:333–345, 2014.

[44] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[45] E. R. Hruschka, R. J. G. B. Campello, A. A. Freitas, and A. C. P. L. F. de Carvalho. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(2):133–155, 2009.

[46] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, Mar. 2002.

[47] M. Hu, T. Wu, and J. D. Weir. An intelligent augmentation of particle swarm optimization with multiple adaptive methods. *Information Sciences*, 213:68–83, 2012.

[48] M. Hu, T. Wu, and J. D. Weir. An adaptive particle swarm optimization with multiple adaptive methods. *IEEE Transactions on Evolutionary Computation*, 17(5):705–720, Oct. 2013.

[49] Y. S. Huang and C. Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):90–94, Jan. 1995.

[50] Y.-S. Hwang and S.-Y. Bang. A neural network model APC-III and its application to unconstrained handwritten digit recognition. In *Proc. of International Conference on Neural Information Processing*, pages 1500–1505, 1994.

[51] Y.-S. Hwang and S.-Y. Bang. An efficient method to construct a radial basis function neural network classifier and its application to unconstrained handwritten digit recognition. In *Proc. of International Conference on Pattern Recognition*, volume 4, pages 640–644, Aug. 1996.

[52] Y.-S. Hwang and S.-Y. Bang. An efficient method to construct a radial basis function neural network classifier. *Neural Networks*, 10(8):1495–1503, 1997.

[53] C. Igel and M. Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86(6):317–321, 2003.

[54] T. Ince, S. Kiranyaz, and M. Gabbouj. Evolutionary RBF classifier for polarimetric SAR images. *Expert Systems with Applications*, 39(5):4710–4717, 2012.

[55] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer for dynamic optimization problems. In *Proc. of Evo Workshops, Applications of Evolutionary Computation*, volume 3005 of *Lecture Notes in Computer Science*, pages 513–524. Springer, Apr. 2004.

[56] B. Jarboui, M. Cheikh, P. Siarry, and A. Rebai. Combinatorial particle swarm optimization (CPSO) for partitional clustering problem. *Applied Mathematics and Computation*, 192(2):337–345, 2007.

[57] V. Kadirkamanathan, K. Selvarajah, and P. J. Fleming. Stability analysis of the particle dynamics in particle swarm optimizer. *IEEE Transactions on Evolutionary Computation*, 10(3):245 – 255, Jun. 2006.

[58] J. Kennedy. *Particle Swarm Optimization*, pages 760 – 766. Springer US, Boston, MA, USA, 2010.

[59] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. of IEEE International Conference on Neural Networks*, pages 1942 – 1948, Nov. 1995.

[60] S. Kiranyaz, M. Gabbouj, J. Pulkkinen, T. Ince, and K. Meissner. Classification and retrieval on macroinvertebrate image databases using evolutionary RBF neural networks. In *Proc. of International Workshop on Advanced Image Technology*, Jan. 2010.

[61] S. Kiranyaz, M. Gabbouj, J. Pulkkinen, T. Ince, and K. Meissner. Network of evolutionary binary classifiers for classification and retrieval in macroinvertebrate databases. In *Proc. of IEEE International Conference on Image Processing*, pages 2257 – 2260, Sep. 2010.

[62] S. Kiranyaz, T. Ince, and M. Gabbouj. Stochastic approximation driven particle swarm optimization with simultaneous perturbation – who will guide the guide? *Applied Soft Computing*, 11(2):2334 – 2347, 2011.

[63] S. Kiranyaz, T. Ince, and M. Gabbouj. *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Springer Verlag Berlin Heidelberg, Germany, 2014.

[64] S. Kiranyaz, T. Ince, J. Pulkkinen, and M. Gabbouj. Classification of Holter registers by dynamic clustering using multi-dimensional particle swarm optimization. In *Proc. of Annual International Conference of IEEE Engineering in Medicine and Biology Society*, pages 4695 – 4698, Sep. 2010.

[65] S. Kiranyaz, T. Ince, S. Uhlmann, and M. Gabbouj. Collective network of binary classifier framework for polarimetric SAR image classification: An evolutionary approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 42(4):1169 – 1186, Aug. 2012.

[66] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj. Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. *Neural Networks*, 22(10):1448 – 1462, 2009.

[67] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj. Fractional particle swarm optimization in multidimensional search space. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(2):298 – 319, 2010.

[68] S. Kiranyaz, T. Mäkinen, and M. Gabbouj. Dynamic and scalable audio classification by collective network of binary classifiers framework: An evolutionary approach. *Neural Networks*, 34:80 – 95, 2012.

[69] S. Kiranyaz, S. Uhlmann, T. Ince, and M. Gabbouj. Perceptual dominant color extraction by multi-dimensional particle swarm optimization. *EURASIP Journal on Advances in Signal Processing*, 2009(451638), 2009.

[70] S. Kiranyaz, S. Uhlmann, J. Pulkkinen, T. Ince, and M. Gabbouj. Incremental evolution of collective network of binary classifier for content-based image classification and retrieval. In *Proc. of International Conference on Innovations in Information Technology*, pages 232 – 237, Apr. 2011.

[71] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671 – 680, May 1983.

[72] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226 – 239, Mar. 1998.

[73] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097 – 1105. Curran Associates, Inc., 2012.

[74] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1), Feb. 1956.

[75] A. M. Leite da Silva, L. S. Rezende, L. M. Honorio, and L. A. F. Manso. Performance comparison of metaheuristics to solve the multi-stage transmission expansion planning problem. *IET Generation, Transmission Distribution*, 5(3):360 – 367, Mar. 2011.

[76] R. Li, B. Bhanu, and A. Dong. Coevolutionary feature synthesized EM algorithm for image retrieval. In *Proc. of ACM International Conference on Multimedia*, pages 696 – 705, Nov. 2005.

[77] R. Li, B. Bhanu, and A. Dong. Feature synthesized EM algorithm for image retrieval. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 4(2):10:1 – 10:24, May 2008.

[78] X. Li, J. Branke, and T. Blackwell. Particle swarm with speciation and adaptation in a dynamic environment. In *Proc. of Annual Conference on Genetic and Evolutionary Computation*, pages 51 – 58, New York, NY, USA, Jul. 2006. ACM.

[79] R.-J. Lian. Adaptive self-organizing fuzzy sliding-mode radial basis-function neural-network controller for robotic systems. *IEEE Transactions on Industrial Electronics*, 61(3):1493 – 1503, Mar. 2014.

[80] Y. Lin and B. Bhanu. Evolutionary feature synthesis for object recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 35(2):156 – 171, 2005.

[81] D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[82] T. Mäkinen, S. Kiranyaz, J. Raitoharju, and M. Gabbouj. An evolutionary feature synthesis approach for content-based audio retrieval. *EURASIP Journal on Audio, Speech, and Music Processing*, 2012(1):1 – 23, Sep. 2012.

[83] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural Networks, Advances in Neural Networks Research: IJCNN'07*, 21(2–3):427–436, 2008.

[84] R. Mendes and A. S. Mohais. DynDE: a differential evolution for dynamic optimization problems. In *Proc. of IEEE Congress on Evolutionary Computation*, volume 3, pages 2808–2815, Sep. 2005.

[85] R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Springer Berlin Heidelberg, 1983.

[86] G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

[87] J. Moody and C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[88] I. Moser and T. Hendtlass. A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 252–259, Sep. 2007.

[89] W. D. Mulder, S. Bethard, and M.-F. Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, 2015.

[90] R. Neruda and P. Kudová. Learning methods for radial basis function networks. *Future Generation Computer Systems*, 21(7):1131–1142, 2005.

[91] G. Neumann, J. Swan, M. Harman, and J. A. Clark. The executable experimental template pattern for the systematic comparison of metaheuristics: Extended abstract. In *Proc. of Companion Publication of Annual Conference on Genetic and Evolutionary Computation*, pages 1427–1430, New York, NY, USA, 2014. ACM.

[92] A. D. Niros and G. E. Tsekouras. A novel training algorithm for RBF neural network using a hybrid fuzzy clustering approach. *Fuzzy Sets and Systems*, 193:62–84, 2012.

[93] M. Omran, A. Salman, and A. P. Engelbrecht. Image classification using particle swarm optimization. In *Proc. of Asia-Pacific conference on simulated evolution and learning*, pages 370–374, Nov. 2002.

[94] M. Omran, A. Salman, and A. P. Engelbrecht. Dynamic clustering using particle swarm optimization with application in image segmentation. *Pattern Analysis and Applications*, 8(4):332–344, Feb. 2006.

[95] F. Pachet and A. Zils. Evolving automatically high-level music descriptors from acoustic signals. In U. K. Wiil, editor, *International Symposium on Computer Music Modeling and Retrieval, Revised Papers*, pages 42–53. Springer Berlin Heidelberg, 2004.

[96] R. Poli. Mean and variance of the sampling distribution of particle swarm optimizers during stagnation. *IEEE Transactions on Evolutionary Computation*, 13(4):712–721, Aug. 2009.

[97] R. Polikar, L. Upda, S. S. Upda, and V. Honavar. Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 31(4):497–508, Nov. 2001.

[98] J. Pulkkinen, S. Kiranyaz, and M. Gabbouj. Dynamic multi-swarm particle swarm optimization with fractional global best formation. In *Proc. of Finnish Artificial Intelligence Conference*, pages 52–59, Aug. 2008.

[99] O. Ritthoff, R. Klinkenberg, S. Fischer, and I. Mierswa. A hybrid approach to feature selection and generation using an evolutionary algorithm. In *Proc. of UK Workshop on Computational Intelligence*, pages 147–154, Feb. 2002.

[100] K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11):2210–2239, 1998.

[101] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[102] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[103] F. Schwenker, H. A. Kestler, and G. Palm. Three learning phases for radial-basis-function networks. *Neural Networks*, 14(4–5):439–458, 2001.

[104] W. Shen, X. Guo, C. Wu, and D. Wu. Forecasting stock indices using radial basis function neural networks optimized by artificial fish swarm algorithm. *Knowledge-Based Systems*, 24(3):378–385, 2011.

[105] J. Sherrah, R. E. Bogner, and A. Bouzerdoum. Automatic selection of features for classification using genetic programming. In *Proc. of Australian and New Zealand Conference on Intelligent Information Systems*, pages 284–287, Nov. 1996.

[106] J. Sherrah, R. E. Bogner, and A. Bouzerdoum. The evolutionary pre-processor: Automatic feature extraction for supervised classification using genetic programming. In *Proc. of Conference on Genetic Programming*, pages 304–312, Jul. 1997.

[107] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 69–73, May 1998.

[108] J. C. Spall. Stochastic optimization. In J. Gentle, W. Härdle, and Y. Mori, editors, *Handbook of Computational Statistics: Concepts and Methods*, chapter 7, pages 173–201. Springer Verlag Heidelberg, 2nd edition, 2012.

[109] D. Steinley. K-means clustering: A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59:1–34, May 2006.

[110] M. R. Tanweer, S. Suresh, and N. Sundararajan. Self regulating particle swarm optimization algorithm. *Information Sciences, Innovative Applications of Artificial Neural Networks in Engineering*, 294:182–202, 2015.

[111] S. Uhlmann, S. Kiranyaz, M. Gabbouj, and T. Ince. Incremental evolution of collective network of binary classifier for polarimetric SAR image classification. In *Proc. of IEEE International Conference on Image Processing*, pages 177–180, Sep. 2011.

[112] Z. Uykan, C. Guzelis, M. E. Celebi, and H. N. Koivo. Analysis of input-output clustering for determining centers of RBFN. *IEEE Transactions on Neural Networks*, 11(4):851–858, Jul. 2000.

[113] Z. Uykan and H. N. Koivo. Analysis of augmented-input-layer RBFNN. *IEEE Transactions on Neural Networks*, 16(2):364–369, Mar. 2005.

[114] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, Pretoria, South Africa, 2002.

[115] F. van den Bergh and A. P. Engelbrecht. A new locally convergent particle swarm optimiser. In *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, volume 3, Oct. 2002.

[116] F. van den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937–971, 2006.

[117] F. van den Bergh and A. P. Engelbrecht. A convergence proof for the particle swarm optimiser. *Fundamenta Informaticae*, 105(4):341–374, Dec. 2010.

[118] L. Vendramin, R. J. G. B. Campello, and E. R. Hruschka. Relative clustering validity criteria: A comparative overview. *Statistical Analysis and Data Mining*, 3(4):209–235, Aug. 2010.

[119] D. Wettschereck and T. G. Dietterich. Improving the performance of radial basis function networks by learning center locations. In *Proc. of International Conference on Neural Information Processing Systems*, pages 1133–1140, 1992.

[120] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.

[121] J. R. Woodward. GA or GP? that is not the question. In *Proc. of IEEE Congress on Evolutionary Computation*, volume 2, pages 1056–1063, Dec. 2003.

[122] R. Xu, J. Xu, and D. C. Wunsch. Clustering with differential evolution particle swarm optimization. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 1–8, Jul. 2010.

[123] R. Xu, J. Xu, and D. C. Wunsch. A comparison study of validity indices on swarm-intelligence-based clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 42(4):1243–1256, 2012.

[124] H. Yu, T. Xie, S. Paszczynski, and B. M. Wilamowski. Advantages of radial basis function networks for dynamic system design. *IEEE Transactions on Industrial Electronics*, 58(12):5438–5450, Dec. 2011.

[125] J. Yu and B. Bhanu. Evolutionary feature synthesis for facial expression recognition. *Pattern Recognition Letters*, 27(11):1289–1298, Aug. 2006.

[126] Z. H. Zhan, J. Zhang, Y. Li, and Y. H. Shi. Orthogonal learning particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 15(6):832–847, Dec. 2011.

[127] B. F. Zhang, J. S. Su, and X. Xu. A class-incremental learning method for multi-class support vector machines in text classification. In *Proc. of International Conference on Machine Learning and Cybernetics*, pages 2581–2585, Aug. 2006.

Errata for Publications

- Publication I: Table 1, step 3.1.1.: Eq. (1) should be Eq. (2), steps 3.4.1.1. and 3.4.1.3.: Eq. (2) should be Eq. (3).
- Publication II: Table 2 illustrates the architecture space defined by $R_{min} = \{N_i, 1, 2\}$ and $R_{max} = \{N_i, 16, 2\}$. Only the dimensions 0 and 8-16 belong to the architecture space used in the experiments.
- Publication II: Table 7: the original ANMRR and AP values for *Corel_10* database with 7 sub-features (i.e., 55.81 and 42.15) should be switched.
- Publication III: Page 3647, just under Fig. 4: $K = N$. Fig. 5 caption: "A sample query...".
- Publication IV: Eq. (5) and Eq. (16) should be given as Eq. (3.4) and Eq. (3.6) in this thesis.
- Publication IV: Page 2217, under Eq. (6): "for each dimensional component except $xd_a(t)$" should be "in each dimension except $xd_a(t)$ for each dimensional component j". Same page, above Eq. (8): "in iteration $t + 1$" should be "in iteration t".
- Publication V: Sect. 3.4: "in that dimensionality, t" should be "in that dimensionality, d_a". Sect 3.5., item 2: "a bit table with b_{min} rows" should be "a bit table with C rows". Fig. 2: $xx_a[6]$ and $xx_{a,j}[6]$ should be x_a^6 and $x_{a,j}^6$. Eq. 6: $f(\mathbf{x}_{p,j}^{d_p}(t))$ should be $f(\mathbf{x}_p^{d_p}(t))$.
- Publication VI: Tables 2, 4, 8: CC_30 should be C_30.
- Publication VII: Page 2463: $\mu_{a,j}$ should be $\mu_{p,j}$.

Publications

Publication I

Turker Ince, Serkan Kiranyaz, Jenni Pulkkinen, Moncef Gabbouj, "Evaluation of Global and Local Training Techniques over Feed-Forward Neural Network Architecture Spaces for Computer-aided Medical Diagnosis," *Expert Systems with Applications*, vol. 37, no. 12, pp. 8450 – 8461, Dec. 2010.

Expert Systems with Applications 37 (2010) 8450–8461

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Evaluation of global and local training techniques over feed-forward neural network architecture spaces for computer-aided medical diagnosis

Turker Ince [a,*], Serkan Kiranyaz [b], Jenni Pulkkinen [b], Moncef Gabbouj [b]

[a] Izmir University of Economics, Faculty of Engineering and Computer Science, Izmir, Turkey
[b] Tampere University of Technology, Tampere, Finland

ARTICLE INFO

Keywords:
Artificial neural networks
Backpropagation
Particle swarm optimization

ABSTRACT

In this paper, we investigate the performance of global vs. local techniques applied to the training of neural network classifiers for solving medical diagnosis problems. The presented methodology of the investigation involves systematic and exhaustive evaluation of the classifier performance over a neural network architecture space and with respect to training depth for a particular problem. In this study, the architecture space is defined over feed-forward, fully-connected artificial neural networks (ANNs) which have been widely used in computer-aided decision support systems in medical domain, and for which two popular neural network training methods are explored: conventional backpropagation (BP) and particle swarm optimization (PSO). Both training techniques are compared in terms of classification performance over three medical diagnosis problems (*breast cancer*, *heart disease*, and *diabetes*) from *Proben1* benchmark dataset and computational and architectural analysis are performed for an extensive assessment. The results clearly demonstrate that it is not possible to compare and evaluate the performance of the two algorithms over a single network and with a fixed set of training parameters, as most of the earlier work in this field has been carried out, since training and test classification performances vary significantly and depend directly on the network architecture, the training depth and method used and the available dataset. We, therefore, show that an extensive evaluation method such as the one proposed in this paper is basically needed to obtain a reliable and detailed performance assessment, in that, we can conclude that the PSO algorithm has usually a better generalization ability across the architecture space whereas BP can occasionally provide better training and/or test classification performance for some network configurations. Furthermore, we can in general say that the PSO, as a global training algorithm, is capable of achieving minimum test classification errors regardless of the training depth, i.e. shallow or deep, and its average classification performance shows less variations with respect to network architecture. In terms of computational complexity, BP is in general superior to PSO for the entire architecture space used.

1. Introduction

Artificial neural networks (ANNs) are known as "universal approximators" and "computational models" with particular characteristics such as the ability to learn or adapt, to organize or to generalize data. Based on the literature in medical domain, computer-aided diagnostic systems with embedded artificial intelligence algorithms have increasingly been used to assist medical experts in diagnosing a patient (Lisboa & Taktak, 2006). Due to their automatic (self-adaptive) process and capability to learn complex, nonlinear surfaces among different classes, ANN classifiers have become a popular choice for decision support in medical diagnosis and have been shown to be effective in the clinical domain (Lisboa, 2002). Most of these neural network classifier studies have used feed-forward, fully-connected ANNs, the well-known multilayer perceptrons (MLPs), with the back-propagation (BP) training algorithm (Lisboa & Taktak, 2006). BP is a gradient descent method on an error surface, which has a local search capability. However, such a capability causes it to get trapped into the nearest local minimum, and thus the training outcome becomes entirely dependent on the initial (weight) settings (Kolen & Pollack, 1990). Therefore, a single BP run is in general unreliable, unrepeatable and sub-optimum, especially in the case of a large number of local minima. In practice, the performance of a BP-ANN classifier depends on the particular pattern recognition or classification problem, description of the input space (extracted features), training sample size, and initial settings of the parameters (i.e. weights, learning rate, and momentum). Currently, there are many variants and extensions of BP to address some of these issues, which in-

* Corresponding author. Tel.: +90 2324888509; fax: +90 2324888475.
E-mail address: turker.ince@ieu.edu.tr (T. Ince).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.05.033

clude gradient descent with momentum, scaled conjugate gradient (SCG), resilient propagation (RPROP), BFGS quasi-Newton, and Levenberg–Marquardt (LM) algorithms (Gudise & Venayagamoorthy, 2003). However, the practitioner must still choose the correct parameter settings with the corresponding algorithm for a specific network and a particular problem.

For many real-world applications, which can be formulated as complex, nonlinear, and multi-modal problems, the global search techniques may perform better since they are capable of finding the global optimum solutions; however, this is never guaranteed. As a recent optimization technique, the particle swarm optimization (PSO) was proposed by Kennedy and Eberhart (1995). It is a population based stochastic search and optimization process. PSO originated from computer simulation of the individuals (particles or living organisms) in a bird flock or a fish school which basically show a natural behavior when they search for some target (e.g. food). The goal is, therefore, to converge to the global optimum of some multi-dimensional and possibly nonlinear function or system. In principle, PSO follows the same path as the existing evolutionary algorithms (EAs). In addition to strong global searching ability, the EA family of algorithms have the advantage of being applicable to any type of ANN, feed-forward or not, with any activation function, differentiable or not. Several researchers have successfully applied PSO for training feed-forward (Carvalho & Ludermir, 2007; Meissner, Schmuker, & Schneider, 2006; Yu, Xi, & Wang, 2007) and recurrent ANNs (Hu & Shi, 2004; Settles, Rodebaugh, & Soule, 2003) to solve classification problems.

Many studies for performance evaluation of neural network classifiers using local and global learning algorithms have recently been presented in the literature. In Hosseini, Luo, and Reynolds (2006), the performance of six feed-forward ANN architectures (four single-hidden layer and two double-hidden layer networks) is investigated for electrocardiogram (ECG) signal diagnosis. The double-hidden layer ANN classifier is shown to enhance the ECG signal classification process. In another study (Van den Bergh, 2002), the author compared the training and testing (generalization) performance of PSO-based algorithms in terms of mean square error (MSE) with BP and Genetic Algorithm (GA)-based techniques. In this study, simple one-hidden layer MLPs with a suitable number of hidden units that are determined by an ANN pruning technique for each problem were used to solve a variety of classification problems including three medical diagnosis problems (breast cancer, diabetes, and hepatitis) from the UCI Machine Learning repository (Prechelt, 1994). It is further shown that PSO-based algorithms can achieve a superior learning ability compared to other algorithms in terms of accuracy and speed. BP and PSO-training for MLPs has also been compared and evaluated over nonlinear function approximation in Kolen and Pollack (1990) and surprisingly it has been shown that PSO outperforms BP in terms of computation complexity required to achieve the same MSE level. In Sexton and Dorsey (2000), researchers performed a direct comparison of BP with the genetic algorithm (GA) for training ANNs over 10 real-world benchmark classification problems from Prechelt (1994). In this study, only three feed-forward network architectures, specifically single-hidden layer MLPs with 3, 6, and 12 hidden nodes, were used for all problems. According to the results, the GA performed consistently better than BP in terms of average classification error (CE) for all 10 problems. In a recent study (Mazurowski et al., 2008), two feed-forward ANN training methods, the traditional BP and PSO are compared by training a single hidden layer MLP with three hidden neurons on real clinical data for breast cancer diagnosis and in contradiction with the aforementioned results, it has been claimed that for imbalanced datasets, BP training yields better results than PSO in terms of average classification performance over the test set.

In this paper, in order to clarify such varying or even contradictory results of the previous studies, we first propose an accurate and in-depth performance assessment approach for comparing local and global training methods, namely BP and PSO for MLPs, particularly over medical diagnosis problems. We shall particularly show that the major problem with the aforementioned evaluations is the usage of only one or few architecture(s) with a static training scheme for a given problem. In this case, one can find a unique architecture and a training depth for a particular problem, over which BP or PSO can surpass the other. In other words, it is quite evident that the performance of both BP and PSO may significantly vary with respect to the network architecture, the training depth and parameters used, and even the classification problem for which the training method is applied. Moreover, PSO and particularly BP yield significantly varying network parameters (weights and biases) after each training session and thus require an exhaustive number of training runs in order to determine statistically significant performance measures and accurate evaluations. In the current work, we focus especially on the average and the best performances that a particular training method can achieve whilst considering both training MSE and test classification error (CE) as the performance criteria. In the current assessment scheme, to avoid the bias of the network architecture used, rather than focusing only one or few architectures, we propose that the evaluations shall be performed over an architecture space containing a large variety of ANNs, from the simplest single-layer perceptrons (SLPs) to the MLPs with several hidden layers and neurons. Furthermore, the training depth is another important factor, which particularly affects the performance of local methods such as BP. For instance a shallow training (with a few iterations), BP is likely to yield a high training MSE and a low test classification error, and vice versa for a deep (or over-) training. As a global search method, PSO may or may not exhibit a similar phenomenon, depending on the other factors, such as the network architecture and training dataset used. Therefore, in this paper, both shallow and deep training depths shall be investigated distinctively while evaluating the performances of both methods in terms of training error (MSE over the training set), generalization capability (CE over the test set) and computational complexity level.

The rest of the paper is organized as follows. Section 2 surveys the related work on BP and PSO and their applications for training ANNs over medical diagnosis problems. Section 3 provides description of the proposed methodology to assess the performance of the corresponding local (BP) and global (PSO) methods for training feed-forward ANNs and discuss the results of the extensive set of experiments conducted over three benchmark problems from the field of medical diagnosis. Finally, Section 4 draws some conclusive remarks and discusses topics for future research.

2. Related work

2.1. The standard BP algorithm

Backpropagation (BP) (Rumelhart, Hinton, & Williams, 1986) is a well-known and widely used supervised training technique for multilayer ANNs with application to pattern recognition and classification in many areas including medical diagnostics. After the development of the BP training algorithm, multilayer perceptron networks have become the standard toolbox of neural network research. MLPs are feed-forward networks with one or more layers of nodes between the input and output nodes (Fig. 1). These additional (hidden) layers contain hidden neurons with nonlinear activation functions. Unlike single-layer perceptrons or multilayer nets with linear elements, a three-layer perceptron was proven to be capable of generating arbitrarily complex decision regions and computing any continuous likelihood function required in a classi-

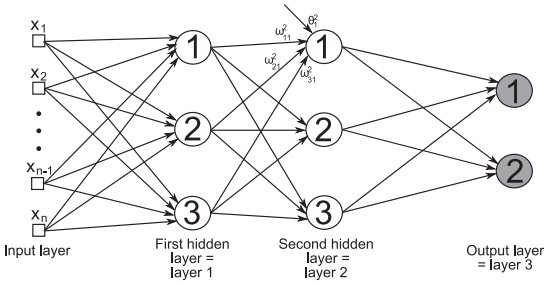


Fig. 1. A sample three-layer perceptron network with continuous valued inputs $\{x_1, \ldots, x_n\}$, two outputs and two layers of hidden units.

fier (Lippmann, 1987). The BP algorithm may be viewed as a generalization of the least-mean-square (LMS) algorithm. It is an iterative gradient search technique designed to minimize the mean square error between the desired and actual net outputs. Backpropagation learning consists of two computational passes through the layers of an MLP: a forward pass which is a feed-forward propagation of input pattern signals through network, and a backward pass during which an error signal is computed at output units and propagated backwards through network. The standard BP algorithm can be summarized as follows:

1. Initialize the weights w_{jk}^l and biases θ_k^l randomly.
2. Feed pattern p to the network and compute the output $y_k^{p,l}$ of each neuron.
3. Calculate the error between the computed output $y_k^{p,o}$ of each output neuron and the desired output t_k^p as $e_k^{p,o} = t_k^p - y_k^{p,o}$.
4. For each neuron k, calculate the local gradients $\frac{\partial E^p}{\partial h_k^l}$, where E^p is the total error energy defined as $E^p = \frac{1}{2}\sum_{k\in o}(e_k^{p,o})^2$ and h_k^l is a uniform symbol for each parameter w_{jk}^l and θ_k^l. The name of the backpropagation algorithm comes from this, as it is necessary to start calculating the local gradients from the output layer and then recursively proceed backwards toward the input layer. Note that smooth (differentiable) nonlinear activation functions must be used for computing the local gradients. The formulas for calculating the local gradients at the output and hidden nodes of MLP can be found in Haykin (1999).
5. Update the parameters as follows:

$$h_k^l(t+1) = h_k^l(t) - \eta\frac{\partial E^p}{\partial h_k^l} \tag{1}$$

where η is the learning-rate parameter.

BP has the advantage of directed search, in that weights are always updated in such a way that minimizes the error. However, there are several aspects, which make the algorithm not guaranteed to be universally useful. Most troublesome is its strict dependency on a learning-rate parameter, which, if not set properly, can either lead to oscillation or indefinitely long training time. Network paralysis might also occur (Back & Tsoi, 1994), i.e. as the ANN trains, the weights tend to be quite large values and the training process can come to a virtual standstill. Furthermore, BP eventually slows down by an order of magnitude for every extra (hidden) layer added to ANN. Above all; BP is just a gradient descent algorithm in the error space, which can be complex and contain many deceiving local minima (multi-modal). Therefore, BP gets most likely trapped into a local minimum, making it entirely dependent on initial (weight) settings. There are many BP variants and extensions trying to address some or all of these problems such as (Haykin, 1999; Joost & Schiffmann, 1998; Riedmiller & Braun, 1993); yet the performance and computational cost of each algorithm varies with respect to the problem at hand; and the question of which ANN architecture (number of layers and interconnections, number of nodes, etc.) should be used for a particular problem still remains unanswered. More detailed information about BP can be found in Chauvin and Rumelhart (1995).

2.2. The basic PSO algorithm

Particle swarm optimization (PSO) was introduced by Kennedy and Eberhart (1995) in 1995 as a population based stochastic search and optimization process. It is originated from a computer simulation of individuals (particles or living organisms) in a bird flock or a fish school (Wilson, 1975), which basically show a natural behavior when they search for some target (e.g. food). The goal is, therefore, to converge to the global optimum of some multi-dimensional and possibly nonlinear function or system. In principle, PSO follows the same path as other evolutionary algorithms (EAs) such as Genetic Algorithm (GA) (Goldberg, 1989), Genetic Programming (GP) (Koza, 1992), Evolutionary Strategies (ES) (Back & Kursawe, 1995), and Evolutionary Programming (EP) (Fayyad, Shapire, Smyth, & Uthurusamy, 1996). In a PSO process, a swarm of particles, each of which represents a potential solution in an optimization problem, navigates through the search space. Particles are initially distributed randomly over the search space and the goal is to converge to the global optimum of a function or a system. Each particle keeps track of its position in the search space and its best solution so far achieved. This is the personal best value (the so-called *pbest* in Kennedy and Eberhart (1995)) and the PSO process also keeps track of the global best solution so far achieved by the swarm with its particle index (the so-called *gbest* in Kennedy and Eberhart (1995)). During their journey with discrete time iterations, the velocity of each particle in the next iteration is affected by the best position of the swarm (best personal position of the particle *gbest* as the social component), the best personal position of the particle (*pbest* as the cognitive component, and its current velocity (the memory term). Both social and cognitive components contribute randomly to the position of the particle in the next iteration. As a stochastic search algorithm in a multi-dimensional (MD) search space, PSO exhibits some major shortcomings similar to the other EAs. A crucial problem for PSO is that parameter variations may result in large performance shifts (Lovberg & Krink, 2002). The second shortcoming is due to the direct link of information flow between particles and gbest, which then "guides" the rest of the swarm and thus resulting in the creation of "similar" particles with a loss of diversity. Hence this phenomenon increases the probability of being trapped in local optima (Riget & Vesterstrom, 2002) and it is the main source of premature convergence especially when the search space is in high dimensions (Van den Bergh, 2002) and the problem to be optimized is multi-modal (Riget et al., 2002). Another reason for the premature convergence is that particles are flown through a single point which is (randomly) determined by *gbest* and *pbest* positions and this point is not even guaranteed to be a local optimum (Van den Bergh, 2002). Various modifications and PSO variants have been proposed in order to address this problem such as (Christopher & Seppi, 2004; Clerc, 1999; Higashi & Iba, 2003; Lovberg, 2002; Riget et al., 2002; Shi & Eberhart, 1998).

In the basic PSO method, a swarm of particles fly through an N-dimensional search space where each particle represents a potential solution to the optimization problem. Each particle a in the swarm $\xi = \{x_1, \ldots, x_a, \ldots, x_S\}$, is represented by the following characteristics:

$x_{a,j}(t)$	jth dimensional component of the position of particle a, at time t
$v_{a,j}(t)$	jth dimensional component of the velocity of particle a, at time t
$y_{a,j}(t)$	jth dimensional component of the personal best (*pbest*) position of particle a, at time t
$\hat{y}_j(t)$	jth dimensional component of the global best (*gbest*) position of swarm, at time t

Let f denote the fitness function to be optimized. Without loss of generality assume that the objective is to find the minimum of f in N-dimensional space. Then the personal best of particle a can be updated in iteration $t+1$ as,

$$y_{a,j}(t+1) = \begin{cases} y_{a,j}(t) & \text{if } f(x_a(t+1)) > f(y_a(t)) \\ x_{a,j}(t+1) & \text{else} \end{cases} \quad j = 1,2,\ldots,N \qquad (2)$$

Since *gbest* is the index of the global best (GB) particle, then $\hat{y}(t) = y_{gbest}(t) = \arg\min_{\forall i \in [1,S]}(f(y_i(t)))$. Then for each iteration in a PSO process, positional updates are performed for each dimension component, $j \in [1,N]$ and for each particle index, $a \in [1,S]$, as follows:

$$\begin{aligned} v_{a,j}(t+1) &= w(t)\,v_{a,j}(t) + c_1 r_{1,j}(t)(y_{a,j}(t) - x_{a,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{a,j}(t)) \\ x_{a,j}(t+1) &= x_{a,j}(t) + v_{a,j}(t+1) \end{aligned} \qquad (3)$$

where $w(t)$ is the inertia weight and c_1, c_2 are the acceleration constants which are initially set to 2 (Shi and Eberhart, 1998). $r_{1,j} \sim U(0,1)$ and $r_{2,j} \sim U(0,1)$ are random variables with uniform distribution. Although the use of inertia weight, $w(t)$, was later added by Shi and Eberhart (1998), into the velocity update equation, it is widely accepted as the basic form of PSO algorithm. A larger value of $w(t)$ favors exploration while a small inertia weight favors exploitation. As originally introduced, $w(t)$ is often linearly decreased from a high value (e.g. 0.9) to a low value (e.g. 0.4) during iterations of a PSO run, which updates the positions of the particles using Eq. (3). Depending on the problem at hand, PSO iterations can be repeated until a specified number of iterations, say *IterNo*, is exceeded, velocity updates become zero, or the desired fitness score is achieved (i.e. $f < \varepsilon_C$). Accordingly, the general pseudo-code of the PSO is presented in Table 1.

Table 1
Pseudo-code for PSO algorithm.

PSO (termination criteria: {IterNo, ε_C, . . .}, V_{max})
1. For $\forall a \in [1,S]$ do:
  1.1. Randomize $x_a(1)$, $v_a(1)$
  1.2. Let $y_a(0) = x_a(1)$
  1.3. Let $\hat{y}(0) = x_a(1)$
2. End For.
3. For $\forall t \in [1, \text{IterNo}]$ do:
  3.1. For $\forall a \in [1,S]$ do:
    3.1.1. Compute $y_a(t)$ using Eq. (1)
    3.1.2. If $(f(y_a(t)) < \min\left(f(\hat{y}(t-1)), \min_{1 \le i < a} f(y_i(t))\right))$ then $gbest = a$ and $\hat{y}(t) = y_a(t)$
  3.2. End For.
  3.3. If any termination criterion is met, then **Stop**.
  3.4. For $\forall a \in [1,S]$ do:
    3.4.1. For $\forall j \in [1,N]$ do:
      3.4.1.1. Compute $v_{a,j}(t+1)$ using Eq. (2)
      3.4.1.2. If$(|v_{a,j}(t+1)| > V_{max})$ then clamp it to $|v_{a,j}(t+1)| = V_{max}$
      3.4.1.3. Compute $x_{a,j}(t+1)$ using Eq. (2)
    3.4.2. End For.
  3.5. End For.
4. End For.

2.3. Applications of BP and PSO over ANNs

For the purpose of medical diagnostics, ANNs have been proposed as decision support tools and have been successfully applied to a wide range of problems such as automated ECG signal diagnosis (Hosseini et al., 2006), electroencephalogram (EEG) waveform classification (Haselsteiner & Pfurtscheller, 2000), decision support systems for breast cancer diagnosis (Lisboa & Taktak, 2006), diagnosis of acute myocardial infarction (coronary occlusion) (Baxt, 1990), recognition of tumors in ultrasound images of the eye (Silverman & Noetzel, 1990), diagnosis of acute pulmonary embolism (Tourassi, Floyd, Sostman, & Coleman, 1993), and differential diagnosis of six dermatology (erythamatous–squamous) diseases (West & West, 2000). MLPs trained with the BP algorithm are used in the majority of the techniques proposed for medical decision support systems where the primary purpose is to augment the physician's decision making in a disease or anomaly diagnosis. However, there are several issues that should be addressed when applying ANN-based classifiers in the medical domain. Besides the selection of the network architecture (type of network, number of layers and interconnections, number of nodes, etc.) and settings of training parameters, the effects of limited training data, description of the input space, large number of features, and class imbalance should be considered during the development phase and the subsequent performance evaluation. Interestingly, despite the fact that many new studies using neural network classifiers have recently been proposed for medical decision making with promising results, only few techniques were able to find their way into clinical use mainly due to poor benchmarking and especially the lack of a proper assessment methodology (Lisboa & Taktak, 2006). One important implication is the need of more extensive and rigorous methodologies for evaluating neural network systems used in medical diagnosis.

Several researchers have successfully applied PSO to train feed-forward and recurrent ANNs that are used in classification problems, some of which are from the medical field. One of the advantages of PSO-based neural network training is that any proper objective function which is more relevant to a particular problem can be chosen and PSO is applicable to any type of ANN, with any activation function. In Eberhart and Hu (1999), a PSO was applied to evolve (train) a feed-forward neural network, which would distinguish between normal subjects and those with tremor (Parkinson's disease or essential tremor) and encouraging results were obtained. Another comparative study over MLPs, which investigates training methods based on many PSO variants, such as multi-start PSO, guaranteed convergence PSO, the traditional BP method, and GA-based techniques was presented in Van den Bergh (2002). This study shows that over the three medical datasets (*breast cancer*, *diabetes*, and *hepatitis*) from *Proben1* (Prechelt, 1994), PSO and its variants achieve a superior training performance compared to the others in terms of classification accuracy on the test set and the training speed. Recently, the effect of class imbalance in the training dataset over the performance of feed-forward ANN classifiers for medical diagnosis was investigated in Mazurowski et al. (2008). As in Van den Bergh (2002), a feed-forward ANN with a single-hidden layer with three neurons was used and BP and PSO were both employed to evaluate the classifier performance using a clinical dataset for breast cancer diagnosis. However, contrary to Van den Bergh (2002), BP training was found to be superior to PSO in terms of (average) test performance.

As one of the most widely applied ANNs, in this study we shall focus on the (supervised) training process of MLP networks. Let N_h^l be the number of hidden neurons in layer l of a MLP with input and output layer sizes N_I and N_O, respectively. The input neurons are merely fan-out units since no processing takes place. Let F be the

activation function applied over the weighted inputs plus a bias, as follows:

$$y_k^{p,l} = F(s_k^{p,l}) \quad \text{where } s_k^{p,l} = \sum_j w_{jk}^{l-1} y_j^{p,l-1} + \theta_k^l \tag{4}$$

where $y_k^{p,l}$ is the output of the kth neuron of the lth hidden/output layer when the pattern p is fed, w_{jk}^{l-1} is the weight from the jth neuron in layer $l-1$ to the kth neuron in layer l, and θ_k^l is the bias value of the kth neuron of the kth hidden/output layer. The training mean squared error, MSE, is formulated as:

$$MSE = \frac{1}{2PN_O} \sum_{p \in T} \sum_{k=1}^{N_O} \left(t_k^p - y_k^{p,O}\right)^2 \tag{5}$$

where t_k^p is the target (desired) output and $y_k^{p,O}$ is the actual output from the kth neuron in the output layer, $l = O$, for pattern p in the training set T with size P. The error measure of MSE is set as an objective (fitness) function of both BP and PSO-training methods to assess their performance under equal training conditions. In PSO, each particle a in the swarm, $\xi = \{x_1, \ldots, x_a, \ldots, x_S\}$, has the positional component formed as, $x_{aj}(t) = \left\{\left\{w_{jk}^0\right\}, \left\{w_{jk}^1\right\}, \{\theta_k^1\}, \left\{w_{jk}^2\right\}, \{\theta_k^2\}, \ldots, \left\{w_{jk}^{O-1}\right\}, \{\theta_k^{O-1}\}, \{\theta_k^O\}\right\}$ where $\left\{w_{jk}^l\right\}$ and $\left\{\theta_k^l\right\}$ represent a potential solution to the problem (the set of weights and biases of layer l). Note that the input layer ($l = 0$) contains only weights whereas the output layer ($l = O$) has only biases. Setting MSE in Eq. (5) as the fitness function enables PSO to perform evolutions of each network parameters within its native process and converge to the best global position found during all iterations.

3. BP vs. PSO: comparative performance evaluation over medical data

3.1. The experimental setup

The architecture space can be defined over a wide range of configurations, i.e. say from a single-layer perceptron (SLP) to complex MLPs with many hidden layers. Suppose, for the sake of simplicity, a range is defined for the number of layers, $[L_{\min}, L_{\max}]$ and another for the number of neurons for each hidden layer l, $\left[N_{\min}^l, N_{\max}^l\right]$. Without loss of generality, assume that the size of both input and output layers is determined by the problem and hence fixed. Consequently, the architecture space can now be defined by only two (MLP) configuration sets, $R_{\min} = \left\{N_I, N_{\min}^1, \ldots, N_{\min}^{L_{\max}-1}, N_O\right\}$ and $R_{\max} = \left\{N_I, N_{\max}^1, \ldots, N_{\max}^{L_{\max}-1}, N_O\right\}$, one for minimum and the other for maximum number of neurons allowed for each layer of a MLP. The size of both arrays is naturally $L_{\max} + 1$ where corresponding entries define the range of neurons possible on the lth hidden layer for all those MLPs, which can have an lth hidden layer. The size of input and output layers, $\{N_I, N_O\}$, is fixed and remains the same for all configurations in the architecture space within which any l-layer MLP can be defined providing that $L_{\min} \leqslant l \leqslant L_{\max}$. $L_{\min} \geqslant 1$ and $L_{\max}$ can be set to any value meaningful for the problem at hand. In this way, all network configurations in the architecture space are enumerated into a hash table with a proper hash function, which basically ranks the networks with respect to their complexity, i.e. associates higher hash indices to networks with higher complexity. The hash function then enumerates all potential MLP configurations into hash indices, starting from the simplest MLP with $L_{\min} - 1$ hidden layers, each of which has a minimum number of neurons given in $R_{\min}$, to the most complex network with $L_{\max} - 1$ hidden layers, each of which has a maximum number of neurons given in $R_{\max}$. Take, for instance, the following configuration sets, $R_{\min} = \{9, 1, 1, 2\}$ and $R_{\max} = \{9, 8, 4, 2\}$, which indicate that $L_{\max} = 3$. If $L_{\min} = 1$ then the hash function enumerates all MLP configurations in the architecture space as shown in Table 2. Note that in this example, the input and output layer sizes are 9 and 2, which are eventually fixed for all MLP configurations. The hash function associates the first index ($d = 0$) with the simplest possible architecture, i.e., a SLP with only input and output layers (9×2). From indices 1 to 8, all configurations belong to 2-layer MLP with a single-hidden layer containing a varying number of neurons between 1 and 8 (as specified in the 2nd entries of arrays $R_{\min}$ and $R_{\max}$). Similarly, for indices 9 and up, 3-layer MLPs are enumerated in which the number of neurons in the 1st and 2nd hidden layers sizes are varied according to the corresponding entries in $R_{\min}$ and $R_{\max}$. Finally, the most complex MLP with the largest number of possible layers and the highest number of neurons is associated with the highest index, $d = 40$. Therefore, all 41 entries in the hash table span the architecture space with respect to the configuration complexity.

Table 2
A sample architecture space for MLP configuration sets $R_{\min} = \{9, 1, 1, 2\}$ and $R_{\max} = \{9, 8, 4, 2\}$.

Index	Configuration	Index	Configuration
0	9×2	21	$9 \times 5 \times 2 \times 2$
1	$9 \times 1 \times 2$	22	$9 \times 6 \times 2 \times 2$
2	$9 \times 2 \times 2$	23	$9 \times 7 \times 2 \times 2$
3	$9 \times 3 \times 2$	24	$9 \times 8 \times 2 \times 2$
4	$9 \times 4 \times 2$	25	$9 \times 1 \times 3 \times 2$
5	$9 \times 5 \times 2$	26	$9 \times 2 \times 3 \times 2$
6	$9 \times 6 \times 2$	27	$9 \times 3 \times 3 \times 2$
7	$9 \times 7 \times 2$	28	$9 \times 4 \times 3 \times 2$
8	$9 \times 8 \times 2$	29	$9 \times 5 \times 3 \times 2$
9	$9 \times 1 \times 1 \times 2$	30	$9 \times 6 \times 3 \times 2$
10	$9 \times 2 \times 1 \times 2$	31	$9 \times 7 \times 3 \times 2$
11	$9 \times 3 \times 1 \times 2$	32	$9 \times 8 \times 3 \times 2$
12	$9 \times 4 \times 1 \times 2$	33	$9 \times 1 \times 4 \times 2$
13	$9 \times 5 \times 1 \times 2$	34	$9 \times 2 \times 4 \times 2$
14	$9 \times 6 \times 1 \times 2$	35	$9 \times 3 \times 4 \times 2$
15	$9 \times 7 \times 1 \times 2$	36	$9 \times 4 \times 4 \times 2$
16	$9 \times 8 \times 1 \times 2$	37	$9 \times 5 \times 4 \times 2$
17	$9 \times 1 \times 2 \times 2$	38	$9 \times 6 \times 4 \times 2$
18	$9 \times 2 \times 2 \times 2$	39	$9 \times 7 \times 4 \times 2$
19	$9 \times 3 \times 2 \times 2$	40	$9 \times 8 \times 4 \times 2$
20	$9 \times 4 \times 2 \times 2$		

The comparative evaluations of both training algorithms were performed using medical diagnosis benchmark dataset from the UCI Machine Learning repository (Prechelt, 1994), which is partitioned into three sets: training, validation and testing. There are several techniques (Amari, Murata, Muller, Finke, & Yang, 1997) to use training and validation sets individually to prevent over-fitting and thus to improve the classification performance in the test data. However, there is no universally effective technique and there are several research articles reporting against the use of the cross-validation technique in the design and training of MLP networks (Amari et al., 1997; Andersen & Martinez, 1999). In this study, for simplicity and to obtain an unbiased performance measure under equal training conditions, the validation and training sets are simply combined to be used for training. From *Proben1* repository (Prechelt, 1994), three benchmark classification problems, breast cancer, heart disease and diabetes, are selected, which were commonly used by the previous studies. These are medical diagnosis problems, which present the following attributes:

- All of them are real-world problems based on medical data from human patients.
- The input and output attributes are similar to those used by a medical doctor.
- Since medical examples are expensive to get, the training sets are quite limited.

We now briefly describe each classification problem next.

3.1.1. Breast cancer

The objective of this data set is to classify breast lumps as either benign or malignant according to microscopic examination of cells that are collected by needle aspiration. There are 699 exemplars of which 458 are benign and 241 are malignant and they are originally partitioned as 350 for training, 175 for validation and 174 for testing. The data set consists of nine input attributes and two output attributes, i.e. each input pattern is described by 9-dimensional vector and there are two possible outcomes of the classifier. It is created at University of Wisconsin Madison by Dr. William Wolberg.

3.1.2. Heart disease

The initial data set consists of 920 exemplars with 35 input attributes, some of which are severely missing. Hence a second data set is composed using the cleanest part of the preceding set, which was created at Cleveland Clinic Foundation by Dr. Robert Detrano. The Cleveland data is called "heartc" in Proben1 repository and contains 303 exemplars but six of them still contain missing data and are hence discarded. The remaining examplars are partitioned as follows: 149 for training, 74 for validation and 74 for testing. There are 13 input and 2 output attributes. The purpose is to predict the presence of a heart disease according to the input attributes.

3.1.3. Diabetes

This dataset is used to predict diabetes diagnosis among Pima Indians. The data is collected from female patients, aged 21 years or older. There are total of 768 exemplars of which 500 are classified as diabetes negative and 268 as diabetes positive. The data set is originally partitioned as 384 for training, 192 for validation and 192 for testing. It consists of eight input attributes and two output attributes.

The input attributes of all data sets are scaled within the range [0, 1] by a linear function. Their output attributes are encoded using a 1-of-c representation using c classes. The winner-takes-all methodology is applied so that the output of the highest activation designates the class. Overall, our experimental setup becomes identical to those used in the previous studies and thus fair comparative evaluations can now be made over the classification error rate of the test data. In all experiments in this section we use the sample architecture space given in Table 2, which has the generalized form as, $R_{\min} = \{N_I, 1, 1, N_O\}$ and $R_{\max} = \{N_I, 8, 4, N_O\}$ containing the compact 1-, 2- or 3-layer MLPs where N_I and N_O, are determined by the number of input and output attributes of the classification problem. For BP, all networks were trained with 500 (shallow training) and with 5000 (deep training) iterations with a low learning rate of 0.02. For PSO-training, in addition to default settings for the standard algorithm parameters as defined in Section 2.3, the number of particles was set to 40 ($S = 40$) and the number of training iterations was set to 200 for shallow and 2000 for deep training case. For all experiments in this section, unless stated otherwise, 100 independent runs are performed for each configuration to compute the error statistics plots for each dataset. We mainly consider two major criteria for the performance assessment: (1) training MSE, which indicates the error minimization achieved by each method and (2) test CE, which is the primary objective of the classifier as it shows the classification accuracy level achieved as well as the generalization capability of each method. Using the corresponding error statistics plots, both criteria shall then be statistically evaluated by considering *on the average* (i.e. mean MSE and CE) and *the best* (i.e. minimum MSE and CE) performances achieved by each method, BP and PSO.

3.2. Evaluations of results with the proposed methodology

In order to perform a comprehensive and systematic assessment of predictive performance of ANN classifiers in medical diagnosis, we apply *exhaustive* BP- and PSO-training for each network configuration in the architecture space, which is defined over MLPs with sigmoid activation functions. In this way we can escape from the bias or possible effect of a particular network over the performance, which was the case of many of the aforementioned studies that were mostly performed using only one or few fixed network architecture(s). Furthermore, to assess the effect of the training depth on both BP and PSO, both shallow and deep training will be applied over every network configuration in the architecture space by setting the number of iterations appropriately.

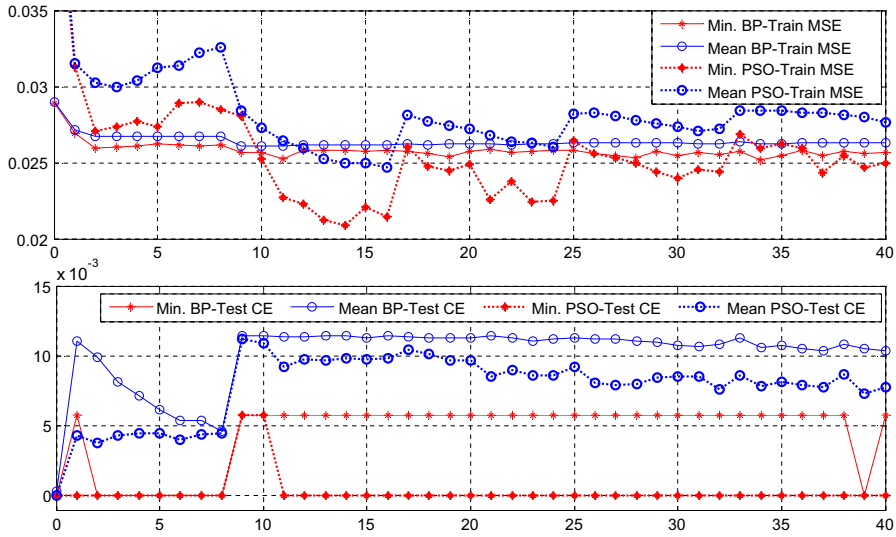


Fig. 2. Train (top) and test (bottom) error statistics vs. hash index plots from *shallow* BP- and PSO-training over the *breast cancer* dataset.

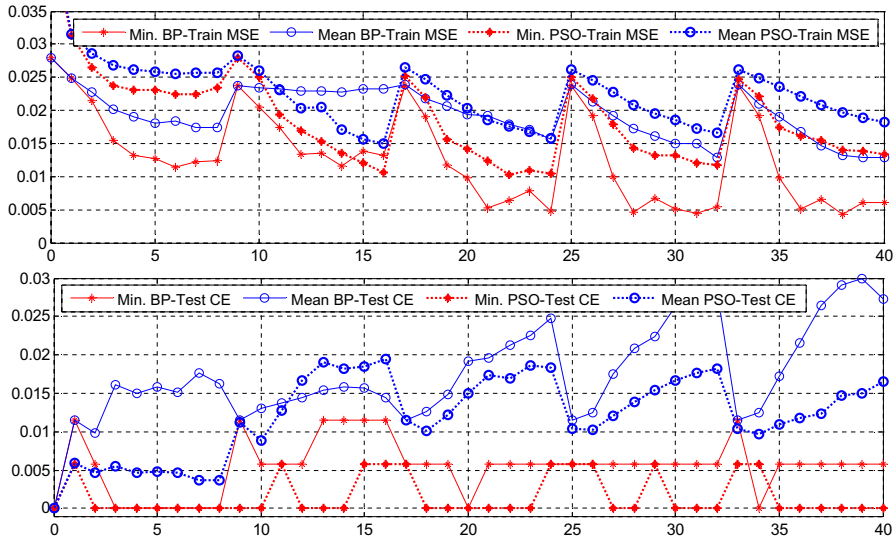


Fig. 3. Train (top) and test (bottom) error statistics vs. hash index plots from *deep* BP- and PSO-training over the *breast cancer* dataset.

Fig. 2 presents the corresponding error statistics plots from the shallow training over the *breast cancer* dataset. BP in general achieves the lowest *average* training MSEs within a narrow variance except few network configurations with the corresponding indices, $d \in [13, 16]$ where PSO slightly surpasses BP. On the other hand, PSO achieves the *best* training performances (i.e. minimum MSEs) over the majority of network configurations except for the compact ones ($d \in [0, 9]$) where BP is consistently more successful. The lowest overall training MSE (both average and minimum) is too achieved by PSO using the configuration with the hash indices $d = 14$ and $d = 16$ (MLPs: $9 \times 6 \times 1 \times 2$ and $9 \times 8 \times 1 \times 2$), respectively. In terms of the classification performances over the test set, the results are consistently in favor of PSO, which performs better than BP with respect to both performance criteria. Particularly, PSO achieved the optimal 0% CE (i.e.100% classification accuracy) as its *best* performance among all networks except for only two networks ($d = 9$ and $d = 10$), whereas BP managed to achieve this only over the compact networks (i.e. $d = 0$ and $d \in [2, 8]$, plus the complex MLP with $d = 39$. Overall, PSO usually demonstrates a better classification performance for the *breast cancer* dataset with the shallow training.

The error statistics plots obtained from deep training of all networks in the architecture space by both methods are shown in Fig. 3. In this case, both BP and PSO achieve lower training MSEs as a natural consequence of the deep- or over-training, and BP in general achieves the lowest *average* training MSEs, particularly on complex networks with two hidden layers but it also surpasses PSO in terms of the *minimum* training MSEs except for only few networks. Due to such over-training, the classification performance of both methods is expected to degrade, which is the case as shown by the bottom plots of Fig. 3. However, the degradation on PSO's performance is not as severe as that of BP. Furthermore, note that there is almost no performance loss in the case of compact networks, due to PSO's global search ability. Particularly for complex networks, a significant performance gap in terms of *average* test CE occurs between the two methods in favor of PSO since BP, as a deterministic local search method, is drastically affected by the over-fitting of the training data and thus exhibits significantly worse average classification performance. This is somewhat true for PSO too; especially its *minimum* CE cannot anymore guarantee 0% CE level for all network configurations.

Fig. 4 presents the corresponding error statistics plots from the *shallow* training over the *heart disease* dataset. Both average and minimum training MSE statistics of both methods vary significantly with respect to the network configuration, making either method better than the other for a given network and error criterion. This is a good example that clearly shows the effect of different network configurations over the performance of each method. Therefore, as discussed earlier about the aforementioned studies in this field, using only one or few architectures for comparative evaluations may favor either method and hence such a static and limited approach is not sufficient to draw reliable conclusions. The lowest training MSE (both average and minimum) is achieved by PSO using the network with the hash index $d = 16$ (MLP: $13 \times 8 \times 1 \times 2$). As in the previous dataset, about the classification performances over the test set, the results are consistently in favor of PSO, which performs better than BP for all networks in the architecture space with respect to both performance criteria.

On contrary to shallow training results, the top plot in Fig. 5 indicates that whenever *deep* training is performed over this dataset, BP surpasses PSO with respect to the training MSEs (both average and minimum) for all networks. Hence due to the over-fitting of training data, the classification performance of BP over the test set is quite degraded whilst no significant performance degradation occurs for PSO. PSO, once again, exhibits its immunity against over-training due to its global search ability and proves to yield the best classification performance over the test set (i.e. well generalization) regardless of the training depth. This is true for both *average* and the *best* performance criteria considered (see red[1] and blue curves of the bottom plot in Fig. 5). PSO achieves the overall best classification performance, ~13% CE, from the three different networks with the corresponding hash indices, $d = 14$, 30 and 39 although no network configuration makes too much difference when *on the average* classification performance is concerned.

[1] For interpretation of color in Fig. 5, the reader is referred to the web version of this article.

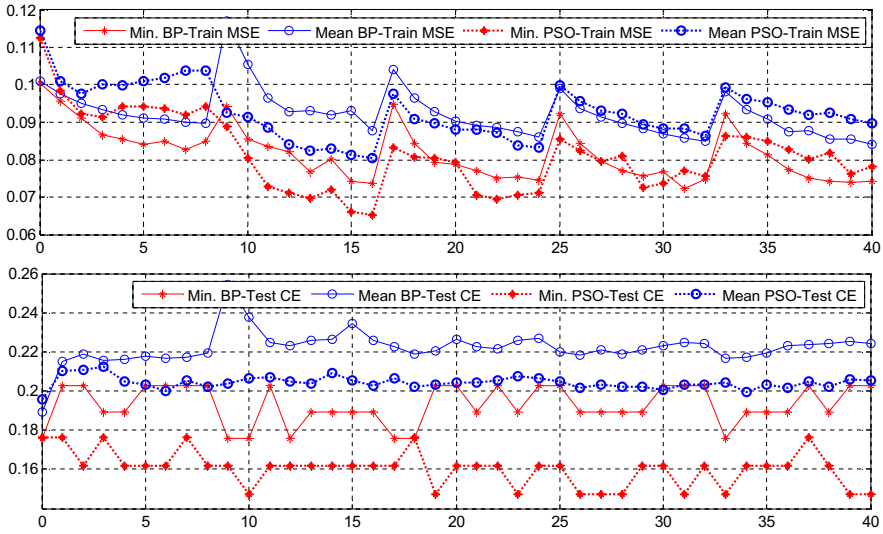


Fig. 4. Train (top) and test (bottom) error statistics vs. hash index plots from *shallow* BP- and PSO-training over the *heart disease* dataset.

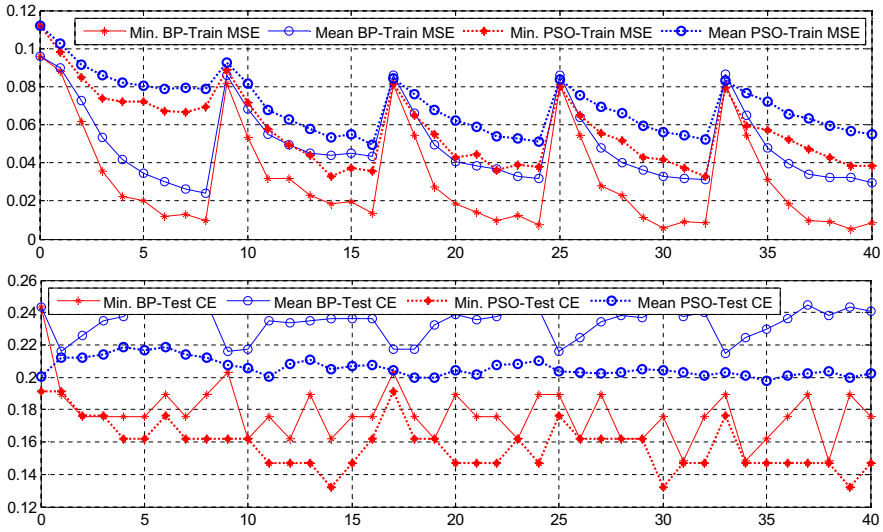


Fig. 5. Train (top) and test (bottom) error statistics vs. hash index plots from *deep* BP- and PSO-training over the *heart disease* dataset.

Fig. 6 presents the corresponding error statistics plots from the *shallow* training over the *diabetes* dataset. Similar comments can be made about the training performance of PSO and BP as in the shallow training experiments over the *heart disease* dataset, i.e. although BP is consistently better than PSO for compact networks, their training performances (minimum and average MSEs) are quite comparable and varying along with the network configuration. In terms of the classification performance over the test set, PSO usually achieves slightly lower CEs but the results are again quite comparable. When the minimum CEs are concerned, from the network with the hash index $d = 16$ (MLP: $8 \times 8 \times 1 \times 2$) PSO achieved a minimum of 17.1% CE that is slightly lower than the 18.8% minimum CE achieved by BP from the network with the hash index $d = 4$ (MLP: $8 \times 4 \times 2$). Finally, according to the error statistics plots in Fig. 7 obtained by deep training over the same dataset, similar comments can be made about both training (MSE) and generalization (test CE) performances of PSO and BP as in the deep training experiments over the *heart disease* dataset, i.e. PSO yields almost the same average training MSE levels and BP significantly reduces both average and minimum MSE levels, as expected. One observation worth mentioning here is that the training and test performances of BP in both deep and shallow training exhibits a large variation with respect to the network configuration used (e.g. compare for instance the mean BP training MSE or test CE

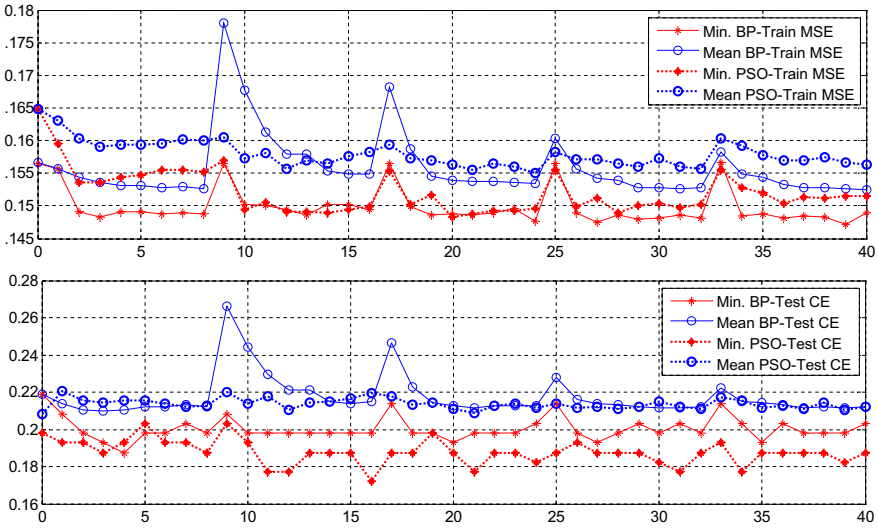


Fig. 6. Train (top) and test (bottom) error statistics vs. hash index plots from *shallow* BP- and PSO-training over the *diabetes* dataset.

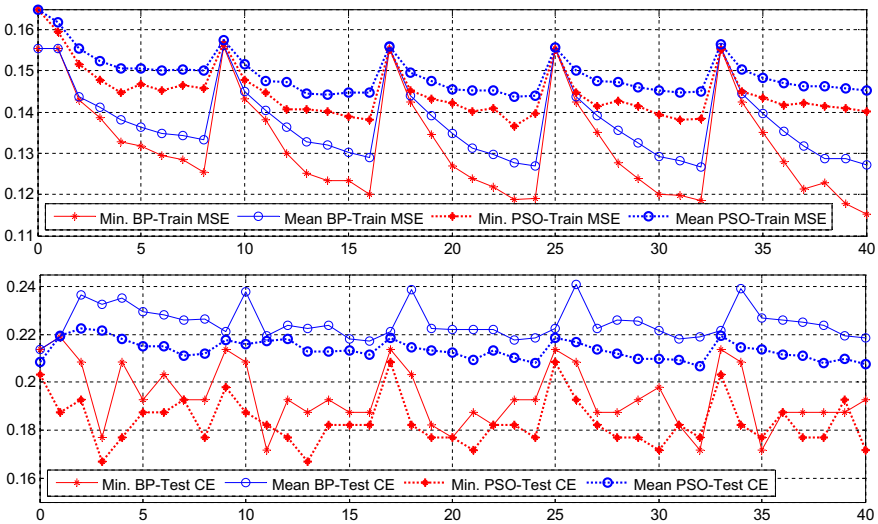


Fig. 7. Train (top) and test (bottom) error statistics vs. hash index plots from *deep* BP- and PSO-training over the *diabetes* dataset.

for $d = 8$ and $d = 9$), whereas the corresponding performance levels of PSO are more stable and usually with a smaller variance, regardless of the network configuration.

The overall test CE statistics of both training techniques (BP and PSO) computed over *all* configurations in selected MLP architecture spaces for each dataset are enlisted in Table 3. We used the following three statistics: minimum *min*, mean μ, and standard deviation σ, respectively, which are computed per training depth (deep and shallow) and separately for the *minimum* and the *mean* test CEs. For all datasets, *minimum* test CE statistics in the table clearly indicates that the overall best classification performances are achieved by PSO-training independent from the training depth. This is also true for the average classification performances with the only exception that in the *heart disease* dataset, minimum of the mean test CEs (corresponding to the best "on the average" test performance) is achieved by BP with the shallow training.

Finally, in order to accomplish the comparative performance evaluations of each method with respect to the variations in the training depth, we have selected a particular network configuration with the hash index $d = 16$, which is a relatively compact and one of the best-performing classifier configuration within the sample architecture space, and we have performed exhaustive training (with 100 runs) for each of the 10 intermediate (training) depths, between the corresponding shallow and deep training, i.e.

Table 3
The overall test CE statistics. The minimum CE statistics are highlighted.

Data set	Training method	Training depth	Min. test CE statistics			Mean test CE statistics		
			min	μ	σ	*min*	μ	σ
Breast cancer	BP	Shallow	**0**	0.0045	0.0024	0.0003	0.0101	**0.0024**
	PSO		**0**	**0.0003**	**0.0013**	**0**	**0.0078**	**0.0024**
	BP	Deep	**0**	0.0055	0.0036	**0**	0.0176	0.0064
	PSO		**0**	0.0015	0.0026	**0**	0.0121	0.0052
Diabetes	BP	Shallow	0.1875	0.2002	**0.0063**	0.2101	0.2175	0.0112
	PSO		0.1719	0.1878	0.0067	0.2079	0.2137	**0.0028**
	BP	Deep	0.1719	0.1941	0.0129	0.2135	0.2246	0.0068
	PSO		**0.1667**	**0.1836**	0.0101	**0.2069**	**0.2135**	0.0040
Heart disease	BP	Shallow	0.1757	0.1928	0.0100	**0.1893**	0.2222	0.0087
	PSO		**0.1471**	0.1603	**0.0092**	0.1957	**0.2043**	**0.0031**
	BP	Deep	0.1486	0.1773	0.0171	0.2150	0.2340	0.0096
	PSO		0.1324	**0.1578**	0.0151	0.1976	0.2060	0.0054

[500, 5000] for BP, and [200, 2000] for PSO. Fig. 8 shows the training MSE and test CE plots vs. training depth for all three medical problems. From the figure, it is clear that both methods reduce training MSE with increasing training depths, as a natural consequence of over-fitting of the training data. On the other hand, PSO achieves lower average and minimum training MSEs for the *breast cancer*, higher for the diabetes and quite similar for the *heart disease* datasets, respectively. The classification performance of PSO shows a strong immunity against variations in training depth and it generally achieves the lowest minimum CEs. For this particular network, either BP or PSO can achieve a better average classification performance than the other, depending on the training depth. Hence this clearly draws the conclusion that the training depth too should be considered while comparing and/or analyzing individual performance of each method.

3.3. Discussion and computational complexity analysis

The overall experimental results, first of all, show that the performance of both methods, BP and PSO, directly depends on the network architecture used and the training depth applied, each of which has varying effects on the two performance criteria employed, minimum and average training MSE and test CE. The former basically shows the search or optimization performance of the method in the training set whereas the latter signifies the generalization ability and the main objective of any ANN training

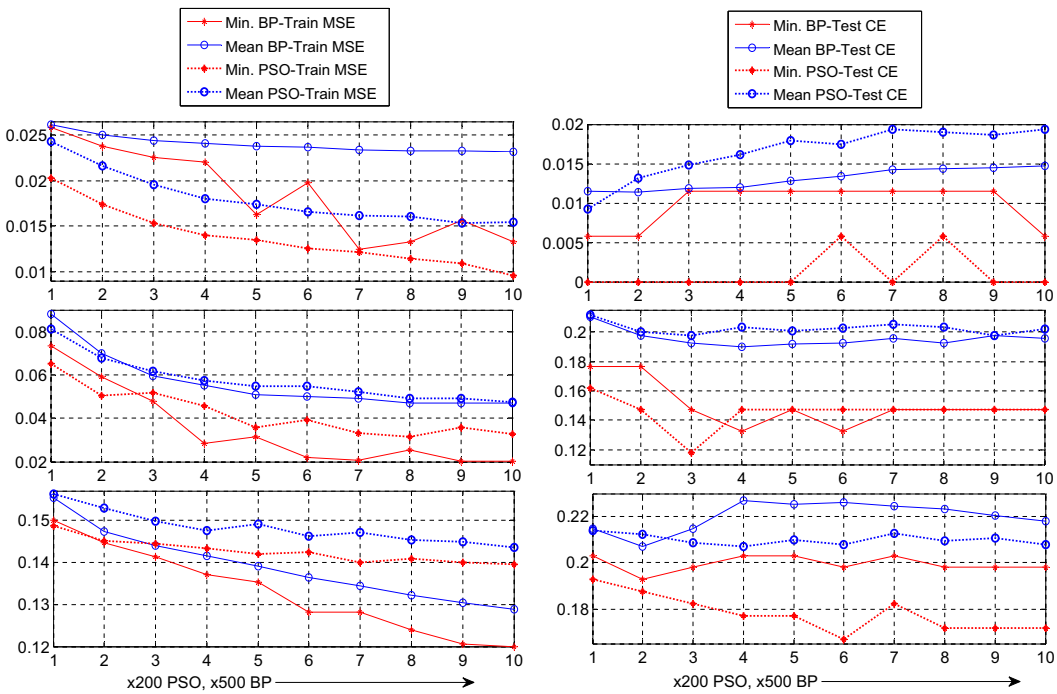


Fig. 8. Error statistics (for network configuration with the hash index $d = 16$) vs. training depth plots using BP and PSO over the *breast cancer* (top), *heart disease* (middle), and *diabetes* (bottom) datasets.

Table 4
Average processing times (in *milliseconds*) of the BP and PSO-training algorithms over the feed-forward architecture space for three medical problems.

Training	Breast cancer		Heart disease		Diabetes	
	BP	PSO	BP	PSO	BP	PSO
Shallow	49.0	236.6	21.3	208.2	114.0	391.3
Deep	436.5	2272.0	217.9	1702.8	670.4	2510.4

method, i.e. the classification performance within the test set. The results justify the use of the proposed assessment technique for both methods, BP and PSO, over the medical datasets. Otherwise, it would be inevitable to obtain varying or even worse contradicting and biased results, which make them all unreliable. For example in Sexton and Dorsey (2000), it is evident that the performance of BP method is significantly underestimated, i.e. the authors of Sexton and Dorsey (2000) reported mean test CE as 3.01% for the *breast cancer*, 24.89% for the *heart disease*, and 29.62% for the *diabetes* datasets where they used only three single-hidden layer MLPs with 3, 6, and 12 hidden nodes, and performed only 10 training runs. In another study (Mazurowski et al., 2008), the authors performed comparative evaluations between the traditional BP and PSO methods over a *breast cancer* dataset and by using only a single-hidden layer MLP with three hidden neurons. With such a limited approach, they made a general conclusion that BP performs better than PSO for *imbalanced* medical datasets. BP perhaps may surpass PSO on that particular MLP, for that dataset and under those specific conditions they employed; however, it is demonstrated in the current work that such an approach is nevertheless not sufficient to draw such a general and decisive conclusion.

Finally, we shall perform computational complexity analysis, which mainly depends on some common properties such as the size of the input and output layers along with the size of the training set (problem specific), and the total number of connections in the network (network complexity). All experiments in this section are performed on a computer with P-IV 3 GHz CPU and 1 GB RAM. BP method consists of one forward and one backward propagation, which requires computation of the error at the output nodes and back-propagating it to the hidden nodes in each hidden layer by calculating the error derivative with respect to weights. Thus the backpropagation of the error (MSE) is computationally the most expensive operation, (it was noticed that backward propagation is 4–8 times more computationally expensive than forward propagation). On the other hand, during a PSO process, at each iteration and for each particle, first the network parameters are extracted from the particle and input vectors are only forward propagated to compute the average error (MSE) at the output layer. Let T be the size of the training data set, S be the swarm size, E be the total number of iterations (epochs) in a PSO run, and μ_t be the average time for a forward propagation. Since the computational complexity is proportional to the total number of forward propagations performed, then the overall computational load for PSO will be in the order $O(SET\mu_t)$. Table 4 presents average training times for three medical benchmark problems by BP and PSO over the sample architecture space given in Table 2. The results clearly indicate that BP is computationally more efficient than PSO with the current parameters of BP and PSO. Since the computational complexity of both methods strictly depends on their parameters (e.g. E for both BP and PSO and S for PSO), it is nevertheless difficult to perform a decisive comparison between them.

4. Conclusions

In this paper, the performance of the two well-known training techniques, BP and PSO, for neural network classifiers over medical datasets was evaluated by using a new assessment methodology, based on a systematic and exhaustive evaluation of the classifier performance over an architecture space and with respect to different training depths. The proposed assessment method can thus evaluate each method's training (error minimization) and test (generalization) performances with respect to different network configurations and training depths whilst considering two statistical criteria, "on the average" and "the best" within an exhaustive number of (training) runs. An extensive experimental study was performed over the three benchmark medical diagnosis problems from *Proben1* repository. In this study, the architecture space was defined over the feed-forward, fully-connected ANNs (the so-called MLPs), which have been widely used in computer-aided decision support systems in medical domain. The experimental results clearly demonstrate that both training and generalization performance of each method depend on the network configuration, the training depth and the particular application at hand. This is basically what has been missing in the related works in this field and that is why there are many variations and even contradictory or mismatching results among them. We strongly believe that this work has important implications for researchers designing neural network classifiers for medical diagnosis systems.

Regarding the performances of both methods, it is concluded first of all that PSO-training has demonstrated relatively better generalization ability across the architecture space whereas BP with deep training always achieved the lowest MSE levels. Furthermore, the classification performance of PSO shows a strong immunity against variations in the training depth, and it generally achieves the lowest minimum CEs. BP, on the other hand, is consistently better than PSO for compact networks, yet the training performances (minimum and average MSEs) of both BP and PSO are quite comparable and usually varying with different network configurations. Contrary to the results published in Mazurowski et al. (2008), we proved that PSO, in both shallow and deep training schemes, usually yields a better classification performance than BP, especially when the minimum CE is taken into account (i.e. the best classification performance). We have further shown statistically that the performance of BP method is drastically underestimated in Sexton and Dorsey (2000) as it can achieve much better classification performance over the entire architecture space used. Finally, as expected, BP is found to be more computationally efficient than PSO.

The proposed method for performance assessment of ANN classifiers can also be used for other types of ANNs with different architecture spaces and in other application areas. This is subject to our future work, which will focus on the application of the proposed assessment methodology for radial basis function (RBF) neural networks over the same benchmark medical datasets. The results will allow us to compare, not only the training methods, BP vs. PSO, but also both types of ANNs, i.e. MLPs vs. RBFs.

References

Amari, S., Murata, N., Muller, K. R., Finke, M., & Yang, H. H. (1997). Asymptotic statistical theory of overtraining and cross-validation. *IEEE Transactions on Neural Networks, 8*(5), 985–996.

Andersen, T., & Martinez, T. (1999). Cross validation and MLP architecture selection. In *Proceedings of international conference on neural networks* (pp. 1614–1619).

Back, A., & Tsoi, A. C. (1994). On the backpropagation algorithm: Paralysis in multilayer perceptrons. In *Proceedings of the fifth Australian conference on neural networks* (pp. 102–104).

Back, T., & Kursawe, F. (1995). Evolutionary algorithms for fuzzy logic: A brief overview. In *Fuzzy logic and soft computing* (pp. 3–10). Singapore: World Scientific.

Baxt, W. G. (1990). Use of an artificial neural network for data analysis in clinical decision-making: The diagnosis of acute coronary occlusion. *Neural Computing, 2*, 480–489.

Carvalho, M., & Ludermir, T. B. (2007). Particle swarm optimization of neural network architectures and weights. In *Proceedings of the 7th international conference on hybrid intelligent systems, Washington DC* (pp. 336–339).

Chauvin, Y., & Rumelhart, D. E. (1995). *Back propagation: Theory, architectures, and applications*. Associates Publishers, Hillsdale, NJ, USA: Lawrence Erlbaum.

Christopher, K. M., & Seppi, K. D. (2004). The Kalman swarm. A new approach to particle motion in swarm optimization. In *Proceedings of the genetic and evolutionary computation conference, GECCO* (pp. 140–150).

Clerc, M. (1999). The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 3, pp. 1951–1957).

Eberhart, R. C. & Hu, X. (1999). Human tremor analysis using particle swarm optimization. In *Proceedings of the congress on evolutionary computation* (pp. 1927–1930).

Fayyad, U. M., Shapire, G. P., Smyth, P., & Uthurusamy, R. (1996). *Advances in knowledge discovery and data mining*. Cambridge, MA: MIT Press.

Goldberg, D. (1989). *Genetic algorithms in search. Optimization and machine learning*. Reading MA: Addison-Wesley. pp. 1–25.

Gudise, V. G., & Venayagamoorthy, G. K. (2003). Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proceedings of the 2003 IEEE swarm intelligence symposium* (pp. 110–117).

Haselsteiner, E., & Pfurtscheller, G. (2000). Using time dependent neural networks for eeg classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering, 8*(4), 457–463.

Haykin, S. (1999). *Neural networks: A comprehensive foundation, 2/E*. New York: Prentice Hall.

Higashi, H., & Iba, H. (2003). Particle swarm optimization with gaussian mutation. In *Proceedings of the IEEE swarm intelligence symposium* (pp. 72–79).

Hosseini, H. G., Luo, D., & Reynolds, K. J. (2006). The comparison of different feed forward neural network architectures for ECG signal diagnosis. *Medical Engineering and Physics, 28*, 372–378.

Hu, X., & Shi, Y. (2004). Recent advances in particle swarm. In *Proceedings of IEEE congress on evolutionary computation* (pp. 90–97).

Joost, M., & Schiffmann, W. (1998). Speeding up backpropagation algorithms by using cross-entropy combined with pattern normalization. *International Journal of Uncertainty, Fuzziness Knowledge-based Systems, 6*(2), 117–126.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of of IEEE international conference on neural networks, Perth, Australia* (Vol. 4, pp. 1942–1948).

Kolen, J. F., & Pollack, J. B. (1990). Back propagation is sensitive to initial conditions. *Complex Systems, 4*, 860–867.

Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Massachussetts: MIT Press, Cambridge.

Lippmann, R. P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*(1987), 4–22.

Lisboa, P. J. (2002). A review of evidence of health benefit from artificial neural networks in medical intervention. *Neural Networks, 15*, 11–39.

Lisboa, P. J., & Taktak, A. F. G. (2006). The use of artificial neural networks in decision support in cancer: A systematic review. *Neural Networks, 19*(4), 408–415.

Lovberg, M. (2002). *Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality*. MSc thesis, Department of Computer Science, University of Aarhus, Denmark.

Lovberg, M. & Krink, T. (2002). Extending particle swarm optimisers with self-organized criticality. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 2, pp. 1588–1593).

Mazurowski, M. A., Habas, P. A., Zurada, J. M., Lo, J. Y., Baker, J. A., & Tourassi, G. D. (2008). Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural Networks, 21*, 427–436.

Meissner, M., Schmuker, M., & Schneider, G. (2006). Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics, 7*, 125.

Prechelt, L. (1994). *Proben1 – A set of neural network benchmark problems and benchmark rules*. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, Germany.

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *Proceedings of the IEEE international conference on neural networks* (pp. 586–591).

Riget, J., & Vesterstrom, J. S. (2002). *A diversity-guided particle swarm optimizer – The ARPSO*. Technical Report, Department of Computer Science, University of Aarhus.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representation by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition* (pp. 318–362). Cambridge, MA: MIT Press.

Settles, M., Rodebaugh, B., & Soule, T. (2003). Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network. In *Lecture notes in computer science (LNCS) no. 2723: proc. of the genetic and evolutionary computation conference 2003 (GECCO 2003), Chicago, IL, USA* (pp. 151–152).

Sexton, R. S., & Dorsey, R. E. (2000). Reliable classification using neural networks: A genetic algorithm and back propagation comparison. *Decision Support Systems, 30*, 11–22.

Shi, Y., & Eberhart, R. C. (1998). A modified particle swarm optimizer. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 69–73).

Silverman, R. H., & Noetzel, A. S. (1990). Image processing and pattern recognition in ultrasonograms by backpropagation. *Neural Networks, 3*, 593–603.

Tourassi, G. D., Floyd, C. E., Sostman, H. D., & Coleman, R. E. (1993). Acute pulmonary embolism: Artificial neural network approach for diagnosis. *Radiology, 189*, 555–558.

Van den Bergh, F. (2002). *An analysis of particle swarm optimizers*. Ph.D. thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa.

Van den Bergh, F., & Engelbrecht, A. P. (2002). A new locally convergent particle swarm optimizer. In *Proceedings of the IEEE international conference on systems, man, and cybernetics* (pp. 96–101).

West, D., & West, V. (2000). Improving diagnostic accuracy using a hierarchical neural network to model decision subtasks. *International Journal Medical Informatics, 57*(1), 41–55.

Wilson, E. O. (1975). *Sociobiology: The new synthesis*. Cambridge, MA: Belknap Press.

Yu, J., Xi, L., & Wang, S. (2007). An improved particle swarm optimization for evolving feedforward artificial neural networks. *Neural Processing Letters, 26*(3), 217–231.

Publication II

Serkan Kiranyaz, Stefan Uhlmann, Jenni Pulkkinen, Moncef Gabbouj, Turker Ince, "Collective Network of Evolutionary Binary Classifiers for Content-Based Image Retrieval," *IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS2011)*, pp. 147 – 154, Apr. 2011.

Collective Network of Evolutionary Binary Classifiers for Content-Based Image Retrieval

Serkan Kiranyaz, Stefan Uhlmann, Jenni Pulkkinen and Moncef Gabbouj
Dept. of Signal Processing
Tampere University of Technology
Tampere, Finland
{firstname.lastname}@tut.fi

Turker Ince
Faculty of Computer Science
Izmir University of Economics
Izmir, Turkey
Email: turker.ince@ieu.edu.tr

***Abstract*—The content-based image retrieval (CBIR) has been an active research field for which several feature extraction, classification and retrieval techniques have been proposed up to date. However, when the database size grows larger, it is a common fact that the overall retrieval performance significantly deteriorates. In this paper, we propose collective network of (evolutionary) binary classifiers (CNBC) framework to achieve a high retrieval performance even though the training (ground truth) data may not be entirely present from the beginning and thus the system can only be trained incrementally. The CNBC framework basically adopts a "Divide and Conquer" type approach by allocating several networks of binary classifiers (NBCs) to discriminate each class and performs evolutionary search to find the optimal binary classifier (BC) in each NBC. In such an evolution session, the CNBC body can further dynamically adapt itself with each new incoming class/feature set without a full-scale re-training or re-configuration. Both visual and numerical performance evaluations of the proposed framework over benchmark image databases demonstrate its scalability; and a significant performance improvement is achieved over traditional retrieval techniques.**

Keywords- evolutionary classifiers, content-based image retrieval, multi-dimensional particle swarm optimization

I. INTRODUCTION

For content-based image classification and retrieval, the key questions, e.g. 1) how to select certain features so as to achieve highest discrimination over certain classes, 2) how to combine them in the most effective way, 3) which distance metric to apply, 4) how to find the optimal classifier configuration for the classification problem in hand, 5) how to scale/adapt the classifier if large number of classes/features are incrementally introduced and finally, 6) how to train the classifier efficiently to maximize the classification accuracy, still remain unanswered. The current state-of-the-art classifiers such as SVMs, Bayesian, Artificial Neural Networks (ANNs), etc. cannot cope with such requirements since a single classifier, no matter how powerful and well-trained it may be, cannot discriminate efficiently a vast amount of classes, over an indefinitely large set of features. Furthermore, since both classes and features are not static, rather dynamically varying, as a natural consequence of image repositories, static and fixed-structured single classifiers cannot scale such changes without proper configuration updates and a full-scale re-training.

In order to address these problems and hence to maximize the classification accuracy which will in turn boost the retrieval performance, in this paper, we shall focus on a global framework design that embodies a collective networks of evolutionary classifiers. Specifically in this approach, the following objectives will be targeted:

I. *Evolutionary Search*: Seeking for the optimum network architecture among a collection of configurations (the so-called Architecture Space, AS).
II. *Evolutionary Update in the AS*: Keeping only "the best" individual configuration in the AS among indefinite number of evolution runs.
III. *Feature Scalability*: Support for varying number of features. Any feature can be dynamically integrated into the framework without requiring a full-scale initialization and re-evolution.
IV. *Class Scalability*: Support for varying number of classes. Any class can dynamically be inserted into the framework without requiring a re-evolution.
V. *High efficiency* for the evolution (or training) process: Using as compact and simple classifiers as possible in the AS.
VI. *Online (incremental) Evolution*: Continuous online/incremental training (or evolution) sessions can be performed to improve the classification accuracy.
VII. *Parallel processing*: Classifiers can be evolved using several processors working in parallel.

In this way, we shall achieve as compact classifiers as possible, which can be evolved and trained in a much more efficient way than a single but complex classifier, and the optimum classifier for the classification problem in hand can be *searched* with an underlying evolutionary technique, e.g. as in [1]. At a given time, this allows creation and designation of a dedicated classifier for discriminating a certain class type from the others based on a single feature. Each incremental evolution session will "learn" from the current best classifier configurations and can improve them further, possibly as a result of an (incremental) optimization, which may find another configuration in the architecture space (AS) as the "optimal". Moreover, with each incremental evolution, new classes/features can also be introduced which signals the collective classifier network to create new corresponding networks and classifiers within to adapt dynamically to the change. In this way the collective classifier network will be able to dynamically *scale* itself to the indexing requirements of the image database whilst striving for maximizing the classification and retrieval accuracies thanks to the dedicated classifiers within. In order to achieve all these objectives, we adopt a *Divide and Conquer* type of approach, which is based on a novel

framework encapsulating a *network of (evolutionary) binary classifiers* (NBCs). Each NBC is devoted to a unique class and further encapsulates a set of *evolutionary Binary Classifiers* (BCs), each of which is optimally chosen within the AS, discriminating the class of the NBC with a unique feature set (or sub-feature). The optimality *therein* can be set with a user-defined criterion. The proposed *Collective NBC* (CNBC) framework currently supports two common ANN types, the Multi-Layer Perceptrons (MLPs) and the Radial Basis Function (RBF) networks. Besides the exhaustive search with the numerous runs of the Back-Propagation method, the recently proposed multi-dimensional Particle Swarm Optimization (MD-PSO) [2], [1] is used as the primary evolution technique. In the current work, due to space limitations we shall restrict only on the CNBC design with evolutionary MLPs.

The rest of the paper is organized as follows. Section II presents evolutionary artificial neural networks. The proposed CNBC framework along with the evolutionary update mechanism is explained in detail in Section III. Section IV provides an extensive set of classification and retrieval results over two benchmark image repositories along with evaluations of the proposed incremental CNBC evolution. Finally, Section V concludes the paper and discusses topics for future work.

II. Evolutionary Neural Networks

In this section we shall discuss the methodology for achieving the first objective that is the evolutionary search for the optimal classifier configuration. First the evolutionary technique, MD-PSO, will be briefly explained and then its application over feed-forward ANNs (the MLPs) shall be introduced. Finally, an overview for the well known training method, the Back Propagation, which can be used exhaustively to search for the optimal classifier in an AS, will briefly be presented.

A. Multi-Dimensional Particle Swarm Optimization

As the evolutionary method, we shall use the multi-dimensional (MD) extension of the basic PSO (*bPSO*) method, the MD-PSO, recently proposed in [2]. Instead of operating at a fixed dimension *N*, the MD-PSO algorithm is designed to seek both positional and dimensional optima within a dimension range, $\{D_{\min}, D_{\max}\}$. In order to accomplish this, each particle has two sets of components, each of which has been subjected to one of the two independent and consecutive processes. The first one is a regular positional PSO, i.e. the traditional velocity updates and due positional shifts in *N* dimensional search (solution) space. The second one is a dimensional PSO, which allows the particle to navigate through dimensions. Accordingly, each particle keeps track of its last position, velocity and personal best position (*pbest*) in a particular dimension so that when it re-visits the same dimension at a later time, it can perform its regular "positional" update using this information. The dimensional PSO process of each particle may then move the particle to another dimension where it will remember its positional status and will be updated within the positional PSO process at this dimension, and so on. The swarm, on the other hand, keeps track of the *gbest* particle in each dimension, indicating the best (global) position so far achieved. Similarly, the dimensional PSO process of each particle uses its personal best dimension in which the personal best fitness score has so far been achieved. Finally, the swarm keeps track of the global best dimension, *dbest*, among all the personal best dimensions. The *gbest* particle in the *dbest* dimension represents the optimum solution and dimension, respectively.

In a MD-PSO process at time (iteration) *t*, each particle *a* in the swarm with *S* particles, $\xi = \{x_1,..,x_a,..,x_S\}$, is represented by the following characteristics:

$xx_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the position of particle *a*, in dimension $xd_a(t)$

$vx_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the velocity of particle *a*, in dimension $xd_a(t)$

$xy_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the personal best position of particle *a*, in dimension $xd_a(t)$

$gbest(d)$: Global best particle index in dimension *d*

$\hat{xy}_j^d(t)$: j^{th} component (dimension) of the global best position of swarm, in dimension *d*

$xd_a(t)$: Dimension of particle *a*

$vd_a(t)$: Dimensional velocity of particle *a*

$\tilde{xd}_a(t)$: Personal best dimension component of particle *a*

Let *f* denote the fitness function that is to be optimized within a certain dimension range, $\{D_{\min}, D_{\max}\}$. Without loss of generality assume that the objective is to find the minimum of *f* at the optimum dimension within a multi-dimensional search space. Assume that the particle *a* visits (back) the same dimension after *T* iterations (i.e. $xd_a(t) = xd_a(t+T)$), then the personal best position can be updated in iteration $t+T$ as follows,

$$xy_{a,j}^{xd_a(t+T)}(t+T) = \begin{cases} xy_{a,j}^{xd_a(t)}(t) & if\ f(xx_a^{xd_a(t+T)}(t+T)) > f(xy_a^{xd_a(t)}(t)) \\ xx_{a,j}^{xd_a(t+T)}(t+T) & else \end{cases} \quad (1)$$

$j = 1,2,...,xd_a(t+T)$

Furthermore, the personal best dimension of particle *a* can be updated in iteration $t+1$ as follows,

$$\tilde{xd}_a(t+1) = \begin{cases} \tilde{xd}_a(t) & if\ f(xx_a^{xd_a(t+1)}(t+1)) > f(xy_a^{\tilde{xd}_a(t)}(t)) \\ xd_a(t+1) & else \end{cases} \quad (2)$$

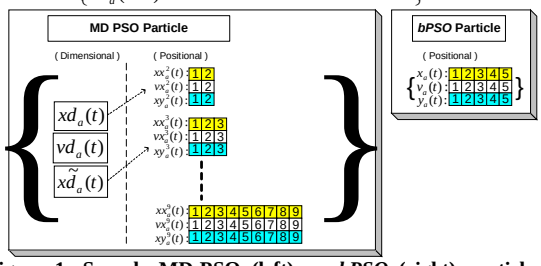


Figure 1: Sample MD-PSO (left) vs. *bPSO* (right) particle structures. For MD-PSO $\{D_{\min}=2, D_{\max}=9\}$ **and at time t,** $xd_a(t)=2$ **and** $\tilde{xd}_a(t)=3$.

Figure 1 shows sample MD-PSO and *bPSO* particles denoted as *a*. Particle *a* in *bPSO* particle is at a (fixed) dimension, N=5, and contains only positional components; whereas in MD-PSO particle *a* contains both positional and dimensional components, respectively. In the figure the dimension range for MD-PSO is given by $\{D_{\min}, D_{\max}\}=\{2, 9\}$, therefore the particle contains 9 sets of positional components. In this example the particle *a* currently resides at dimension 2 ($xd_a(t)=2$); whereas its personal best

dimension is 3 ($\tilde{xd}_a(t) = 3$). Therefore, at time *t* a positional PSO update is first performed over the positional elements, $xx_a^2(t)$ and then the particle may move to another dimension with respect to the dimensional PSO. Recall that each positional element, $xx_a^2(t)$, represents a potential solution in the search space of the problem.

B. MD-PSO for Evolving MLPs

As a stochastic search process in multi-dimensional search space, MD-PSO seeks (near-) optimal networks in an architecture space (AS), which can be defined over any type of ANNs with any properties. All network configurations in the AS are enumerated into a hash table with a proper hash function, which ranks the networks with respect to their complexity, i.e. associates higher hash indices to networks with higher complexity. MD-PSO can then use each index as a unique dimension of the search space where particles can make inter-dimensional navigations to seek an optimum dimension (*dbest*) and the optimum solution on that dimension, $\hat{xy}^{dbest}$. The former corresponds to the optimal architecture and the latter encapsulates the optimum network parameters (connections, weights and biases). Suppose for the sake of simplicity, a range is defined for the minimum and maximum number of layers, $\{L_{\min}, L_{\max}\}$ and number of neurons for the hidden layer *l*, $\{N^l_{\min}, N^l_{\max}\}$. The sizes of both input and output layers are determined by the problem and hence fixed. The AS can then be defined only by two range arrays, $R_{\min} = \{N_i, N^1_{\min},...,N^{L_{\max}-1}_{\min}, N_o\}$ and $R_{\max} = \{N_i, N^1_{\max},...,N^{L_{\max}-1}_{\max}, N_o\}$, one for minimum and the other for the maximum number of neurons allowed for each layer of a MLP. The size of both arrays is naturally $L_{\max}+1$ where the corresponding entries define the range of the l^{th} hidden layer for all those MLPs, which can have an l^{th} hidden layer. The size of the input and output layers, $\{N_i, N_o\}$, is fixed and is the same for all configurations in the AS. $L_{\min} \geq 1$ and $L_{\max}$ can be set to any value meaningful for the problem encountered. The hash function then enumerates all potential MLP configurations into hash indices, starting from the simplest MLP with $L_{\min}-1$ hidden layers, each of which has minimum number of neurons given by $R_{\min}$, to the most complex network with $L_{\max}-1$ hidden layers, each of which has a maximum number of neurons given by $R_{\max}$.

Let N^l_h be the number of hidden neurons in layer *l* of a MLP with input and output layer sizes N_i and N_o, respectively. The input neurons are merely fan-out units since no processing takes place. Let *F* be the activation function applied over the weighted inputs plus a bias, as follows:

$$y_k^{p,l} = F(s_k^{p,l}) \quad where \quad s_k^{p,l} = \sum_j w_{jk}^{l-1} y_j^{p,l-1} + \theta_k^l \qquad (3)$$

where $y_k^{p,l}$ is the output of the k^{th} neuron of the l^{th} hidden/output layer when the pattern *p* is fed, w_{jk}^{l-1} is the weight from the j^{th} neuron in layer *l*-1 to the k^{th} neuron in layer *l*, and θ_k^l is the bias value of the k^{th} neuron of the l^{th} hidden/output layer. The training mean square error, *MSE*, is formulated as follows:

$$MSE = \frac{1}{2PN_o} \sum_{p \in T} \sum_{k=1}^{N_o} \left(t_k^p - y_k^{p,o}\right)^2 \qquad (4)$$

where t_k^p is the target (desired) output and $y_k^{p,o}$ is the actual output from the k^{th} neuron in the output layer, $l=o$, for pattern *p* in the training dataset *T* with size *P*, respectively. At a time *t*, suppose that particle *a*, has the positional component formed as, $xx_a^{xd_a(t)}(t) = \Psi^{xd_a(t)}\left\{\{w^0_{jk}\},\{w^1_{jk}\},\{\theta^1_k\},\{w^2_{jk}\},\{\theta^2_k\},...,\{w^{o-1}_{jk}\},\{\theta^{o-1}_k\},\{\theta^o_k\}\right\}$ where $\{w^l_{jk}\}$ and $\{\theta^l_k\}$ represent the sets of weights and biases of the layer *l* of the MLP configuration, $\Psi^{xd_a(t)}$. Note that the input layer (l=0) contains only weights whereas the output layer (l=o) has only biases. By means of such a direct encoding scheme, particle *a* represents all potential network parameters of the MLP architecture at the dimension (hash index) $xd_a(t)$. As mentioned earlier, the dimension range, $\{D_{\min}, D_{\max}\}$, where MD-PSO particles can make inter-dimensional jumps, is determined by the AS defined. Apart from the regular limits such as (positional) velocity range, $\{V_{\min}, V_{\max}\}$, dimensional velocity range, $\{VD_{\min}, VD_{\max}\}$, the data space can also be limited with some practical range, i.e. $X_{\min} < xx_a^{xd_a(t)}(t) < X_{\max}$. Setting *MSE* in Eq. (4) as the fitness function enables MD-PSO to perform evolutions of both network parameters and architectures within its native process.

Further details and an extensive set of experiments demonstrating the optimality of the networks evolved with respect to several benchmark problems can be found in [1].

C. The Back Propagation Algorithm

Back Propagation (BP) [3] is the most commonly used training technique for feed-forward ANNs. It is a supervised training technique which has been used in pattern recognition and classification problems in many application areas. BP has the advantage of applying directed search and has a local search ability. However, BP is just a gradient descent algorithm in the error space, which can be complex and may contain many deceiving local minima (multi-modal). Therefore, BP gets most likely trapped into a local minimum, making it entirely dependent on the initial (weight) settings. Yet due to its simplicity and relatively lower computational cost, BP can be applied exhaustively over the network architectures of AS with random initializations to find out the optimal architecture for the problem in hand. Since AS is composed of only compact networks, with such an exhaustive application, the probability of finding (converging) to a (near-) optimum solution in the error space is significantly increased.

The BP algorithm can be summarized as follows:

1. Initialize the weights w^l_{jk} and biases θ^l_k randomly.

2. Feed pattern *p* to the network and compute the output $y_k^{p,l}$ of each neuron.

3. Calculate the error between the computed output $y_k^{p,o}$ of each output neuron and the desired output t_k^p as $e_k^{p,o} = t_k^p - y_k^{p,o}$.

This work was supported by the Academy of Finland, project No. 213462 (Finnish Centre of Excellence Program (2006 - 2011)

4. For each neuron *k*, calculate the partial derivatives $\frac{\partial E^p}{\partial h_k^l}$, where E^p is the total error energy defined as $E^p = \frac{1}{2}\sum_{k\in o}(e_k^{p,o})^2$ and h_k^l is a uniform symbol for each parameter $w_{jk}^l, \theta_k^l, \mu_k$ and σ_k.

5. Update the parameters as follows:

$$h_k^l(t+1) = h_k^k(t) - \eta \frac{\partial E^p}{\partial h_k^k} \quad (5)$$

where η is the learning rate parameter.

6. Repeat steps 2-5 until some stopping criteria is reached.

In the above realization of the BP algorithm the network parameters are updated after every training sample. This is called the *online* or *sequential* mode. The other possibility is the *batch* mode, where all the training samples are first presented to the network and then the parameters are adjusted so that the total training error is minimized. The *sequential* mode is often favored over the *batch* mode as it requires less storage space. Moreover, the *sequential* mode is less likely to get trapped in a local minimum as updates at every training sample make the search stochastic in nature. Hence *sequential* BP mode is used for MLP training/evolution. Further details about BP training can be found in [3] and are hence skipped.

III. The CNBC Framework

This section describes in detail the proposed framework: Collective Network of (Evolutionary) Binary Classifiers, the CNBC, which uses the training dataset to configure its internal structure and to evolve its binary classifiers (BCs) individually. Before going into details of CNBC, the evolutionary update mechanism, which keeps only the best networks within the AS of each BC will be detailed next.

A. Evolutionary Update in the Architecture Space

Since the evolutionary technique, MD PSO, is a stochastic optimization method, in order to improve the probability of convergence to the global optimum, several evolutionary runs can be performed. Let N_R be the number of runs and N_C be the number of configurations in the AS. For each run the objective is to find the optimal (the best) classifier within the AS with respect to a pre-defined criterion. Note that along with the best classifier, all other configurations in the AS are also subject to evolution and therefore, they are continuously (re-) trained with each run. So during this ongoing process, between any two consecutive runs, any network configuration can replace the current best one in the AS if it surpasses it. Besides the MD PSO evolutionary search, this is also true for the *exhaustive search*, i.e. each network configuration in the AS is trained by N_R BP runs and the same evolutionary update rule applies. Figure 2 demonstrates an evolutionary update operation over a sample AS containing 5 MLP configurations. The table shows the training MSE which is the criterion used to select the optimal configuration at each run. The best runs for each configurations are highlighted and the best configuration in each run is tagged with '*'. Note that at the end of the three runs, the overall best network with MSE = 0.10 has the configuration: 15x2x2 and thus used as the classifier for any classification task until any other configuration surpasses it in a future run. In this way, each BC configuration in the AS can only *evolve* to a better state, which is the main purpose of the proposed evolutionary update mechanism.

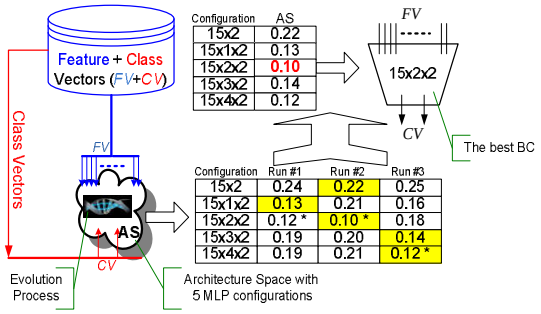


Figure 2: Evolutionary update in a sample AS for MLP configuration arrays $R_{\min} = \{15,1,2\}$ and $R_{\max} = \{15,4,2\}$ where $N_R = 3$ and $N_C = 5$. The best run for each configurations is highlighted and the best configuration in each run is tagged with '*'.

B. Collective Network of Binary Classifiers

1) The Topology

To achieve the third and fourth objectives mentioned earlier, i.e. the scalability with respect to a varying number of classes and features, a novel framework encapsulating a *network of binary classifiers* (NBCs) is developed, where NBCs can *evolve* continuously with the ongoing evolution sessions i.e. using the training dataset which is in practice obtained by cumulating the ground truth data (GTD) from several relevance feedback sessions. Each NBC corresponds to a *unique* image class and shall contain varying number of evolutionary binary classifiers (BCs) in the input layer where each BC performs binary classification using a single (sub-) feature. Therefore, whenever a new feature is extracted, its corresponding BC will be created, evolved (using the available GTD so far), and inserted into each NBC, yet keeping each of the other BCs "as is". On the other hand, whenever an existing feature is removed, the corresponding BC is simply removed from each NBC in the system. In this way *scalability* with respect to any number of features is achieved and the overall system can avoid re-evolutions from scratch.

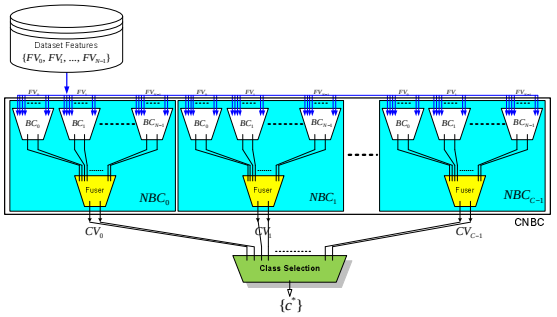


Figure 3: Topology of the proposed CNBC framework with *C* classes and *N* FVs (sub-features).

Each NBC has a "fuser" BC in the output layer, which collects and fuses the binary outputs of all BCs in the input layer

and generates a single binary output, indicating the relevancy of each FV to the NBC's corresponding class.

Furthermore, CNBC is also *scalable* to any number of classes since whenever a new class is defined by the user, a new NBC can simply be created (and evolved) only for this class without requiring any need for change or update the other NBCs unless their performance significantly deteriorates with the introduction of the new class This way the overall system dynamically adapts to varying number of image classes.

As shown in Figure 3, the main idea in this approach is to use as large number of classifiers as necessary, so as to divide a massive learning problem into many NBC units along with the BCs within, and thus prevent the need of using complex classifiers as the performance of both training and evolution processes degrades significantly as the complexity rises due to the *curse of dimensionality*. A major benefit of our approach with respect to efficient training and evolution process is that the configurations in the AS can be kept as *compact* as possible avoiding unfeasibly large storage and training time requirements. This is a significant advantage especially for the training methods performing local search, such as BP since the amount of deceiving local minima is significantly low in the error space for such simple and compact ANNs. Furthermore, when BP is applied exhaustively, the probability of finding the optimum solution is significantly increased.

In order to maximize the classification accuracy, we applied a dedicated class selection technique for CNBC. We used *1-of-n* encoding scheme in all BCs, and the output layer size of all BCs is always two (i.e. $n = 2$). Let $CV_{c,1}$ and $CV_{c,2}$ be the first and second output of the c^{th} BC's class vector (CV). The class selection in *1-of-n* encoding scheme can simply be performed by comparing the individual outputs, e.g. say a positive output if $CV_{c,2} > CV_{c,1}$, and vice versa for negative. This is also true for the fuser BC, the output of which makes the output of its NBC. FVs of each dataset item are fed to each NBC in the CNBC. Each FV is propagated through its corresponding BC in the input layer of the NBC. The outputs of these BCs are then fed to the fuser BC of each NBC to produce all CVs. Finally, the class selection block shown in Figure 3 collects them and selects the positive class(es) of the CNBC as the final outcome. This selection scheme, first of all, differs with respect to the dataset class type, i.e. the dataset can be called as "uni-class", if an item in the dataset can belong to *only* one class, otherwise called as "multi-class". Therefore, in a uni-class dataset there must be *only* one class, the c^*, selected as the positive outcome whereas in a multi-class dataset, there can be one or more NBCs, $\{c^*\}$, with a positive outcome. In the class selection scheme the *winner-takes-all* strategy is utilized. Therefore, for uni-class datasets, the positive class index, c^*, ("the winner") is determined to be the class where the difference $CV_{c,2} - CV_{c,1}$ is maximal, i.e.,

$$c^* = \arg \max_{c \in [0, C-1]} (CV_{c,2} - CV_{c,1}) \qquad (6)$$

In this way the erroneous cases (false negative and false positives) where none or more than one NBC exists with a positive outcome can be properly handled. However, for multi-class datasets the winner takes all strategy can only be applied when no NBC yields a positive outcome, i.e. $CV_{c,2} \le CV_{c,1} \;\; \forall c \in [0, C-1]$, otherwise multiple NBCs with positive outcome may indicate the multiple true-positives and hence cannot be further pruned. As a result, for a multi-class dataset the (set of) positive class indices, $\{c^*\}$, is selected as follows:

$$\{c^*\} = \begin{cases} \arg \max_{c \in [0, C-1]} (CV_{c,2} - CV_{c,1}) & if \;\; CV_{c,2} \le CV_{c,1} \;\; \forall c \in [0, C-1] \\ \{ \arg_{c \in [0, C-1]} (CV_{c,2} > CV_{c,1}) \} & else \end{cases} \qquad (7)$$

2) *Evolution of the CNBC*

The evolution of a subset of the NBCs or the entire CNBC is performed for each NBC individually with a two-phase operation, as illustrated in Figure 4. As explained earlier, using the feature vectors (FVs) and the target class vectors (CVs) of the training dataset, the evolution process of each BC in a NBC is performed within the current architecture space (AS) in order to find the best (optimal) BC configuration with respect to a given criterion (e.g. training/validation mean-square-error (MSE) or classification error (CE)). During the evolution, only NBCs associated with those classes represented in the training dataset are evolved. If the training dataset contains new classes, which do not have a corresponding NBC yet, a new NBC is created for each, and evolved using the training dataset.

In Phase 1, see top of Figure 4, the BCs of each NBC are evolved given an input set of FVs and a target CV. Recall that each CV is associated with a unique NBC. The fuser BCs are not used in this phase. Once an evolution session is over, the AS of each BC is then recorded so as to be used for potential (incremental) evolution sessions in the future.

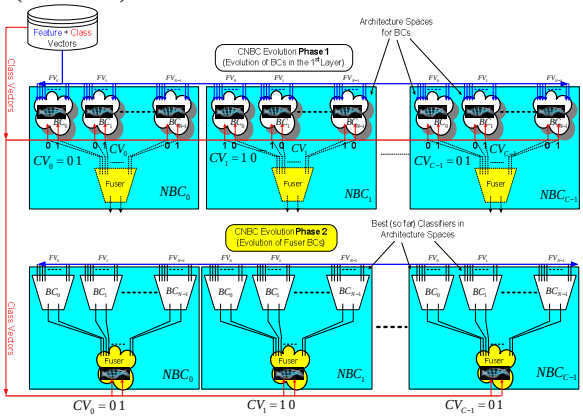


Figure 4: Illustration of the two-phase evolution session over BCs' architecture spaces in each NBC.

Recall that each evolution process may contain several runs and according to the aforementioned evolutionary update rule, the best configuration achieved will be used as the classifier. Hence once the evolution process is completed for all BCs in the input layer (phase 1), the best BC configurations are used to forward propagate all FVs of the items in the training dataset to compose the FV for the fuser BC from their output CVs, so as to evolve the fuser BC in the second phase. Apart from the difference in the generation of the FVs, the evolutionary method (and update) of the fuser BC is same as any other BC has in the

input layer. In this phase, the fuser BC *learns* the *significance* of each individual BC (and the corresponding sub-feature) for the discrimination of that particular class. This can be viewed as the adaptation of the entire feature space to discriminate a specific class in a large dataset, or in other words, as a way of applying an efficient *feature selection* scheme as some FVs may be quite discriminative for some classes whereas others may not and the fuser, if properly evolved and trained, can "weight" each BC (with its FV), accordingly. In this way the usage of each feature (and its BC) shall optimally be "fused" according to their discrimination power of each class. Similarly, each BC in the first layer shall in time learn the significance of individual feature components of the corresponding FV for the discrimination of its class. In short the CNBC, if properly evolved, shall learn the significance (or the discrimination power) of each FV and its individual components.

3) Incremental Evolution of the CNBC

To accomplish yet another major objective, the proposed CNBC framework is designed for continuous "incremental" evolution sessions where each session may further improve the classification performance of each BC using the advantage of the "evolutionary updates". The main difference between the initial and the subsequent incremental evolution sessions is the initialization of the evolution process: the former uses *random* initialization for each configuration in the AS whereas the latter starts from the best parameters found for each classifier configurationin the last AS for each BC. Note that the training dataset used for the incremental evolution sessions may be different from the previous ones, and each session may contain several runs. Thus the evolutionary update rule compares the performance of the last recorded and the *current* (after the run) network over the *current* training dataset. During each incremental evolution phase, existing NBCs are (incrementally) evolved *only if* they cannot accurately classify the training dataset of the new (emerging) classes. In that, an empirical threshold level (e.g. 95%) is used to determine the level of classification accuracy required. The NBCs for the new classes are obviously due for evolution without any such verification.

Consequently, the proposed MD PSO evolutionary technique used for evolving MLP configurations is initialized with the current AS parameters of the network. That is the swarm particles are randomly initialized (as in the initial evolutionary step) except that one of the particles (without loss of generality we assume the first particle with a=0) has its personal best set to the optimal solution found in the previous evolutionary session. For MD PSO evolution over MLPs, this can be expressed as,

$$xy_0^d(0) \leftarrow \Psi^d\left\{\{w_{jk}^0\},\{w_{jk}^1\},\{\theta_k^1\},\{w_{jk}^2\},\{\theta_k^2\},...,\{w_{jk}^{O-1}\},\{\theta_k^{O-1}\},\{\theta_k^O\}\right\} \quad \forall d \in [1, N_C] \tag{8}$$

where $\{w_{jk}^l\}$ and $\{\theta_k^l\}$ represent the sets of weights and biases of the layer l of the MLP network, Ψ^d, which is the d^{th} (MLP) configuration retrieved from the last AS record. It is expected that especially at the early stages of the MD PSO run, the first particle is likely to be the *gbest* particle in every dimension, guiding the swarm towards the last solution otherwise keeping the process independent and unconstrained. Particularly if the training dataset is considerably different in the incremental evolution sessions, it is quite probable that MD PSO can converge to a new solution whilst taking the past solution (experience) into account.

In the alternative evolutionary technique, the exhaustive search via repetitive BP training of each network in the AS, the first step of an incremental training will simply be the initialization of the weights w_{jk}^l and biases θ_k^l with the parameters retrieved from the last AS of that BC. Starting from this as the initial point, and using the current training dataset with the target CVs, the BP algorithm can then perform its gradient descent in the error space to converge to a new solution.

IV. Experimental Results

In this section, we first detail the benchmark datasets used and the feature extraction techniques performed for the extensive set of classification and content-based image retrieval experiments. We then investigate the classification performance of the proposed CNBC framework using different feature combinations and we also compare the results obtained with batch training and incremental evolutions. Finally we shall demonstrate the performance gain in terms of improved retrieval accuracy that can be achieved using the proposed CNBC framework as compared to the traditional (dis-) similarity based retrievals.

A. Database Creation and Feature Extraction

We used MUVIS framework [4], to create and to index the following two image databases by extracting 14 (sub-) features for each.

1) ***Corel_10*** Image Database: There are 1000 medium resolution (384x256 pixels) images obtained from Corel repository [5] covering 10 diverse classes: 1 - *Natives*, 2 - *Beach*, 3 - *Architecture*, 4 - *Bus*, 5 - *Dino Art*, 6 - *Elephant*, 7 - *Flower*, 8 - *Horse*, 9 - *Mountain*, and 10 - *Food*.

2) ***Corel_Caltech_30*** Image Database: There are 4245 images from 30 diverse classes that are obtained from both Corel and Caltech [6] image repositories.

Table 1: 14 Features extracted per MUVIS database.

FV	Feature	Parameters	Dim.
1	HSV Color Histogram	H=6, S=2, V=2	**24**
2		H=8, S=4, V=4	**128**
3	Dominant Color	$N_{DC}^{\max}=6, T_A=2\%, T_S=15$	**27**
4		$N_{DC}^{\max}=8, T_A=2\%, T_S=15$	**35**
5	Color Structure	32 bins	**32**
6		64 bins	**64**
7		128 bins	**128**
8		256 bins	**256**
9		512 bins	**512**
10		1024 bins	**1024**
11	Local Binary Pattern		**16**
12	Gabor	scale=4, orient.=6	**48**
13	Ordinal Co-occurrence	d=3, o=4	**36**
14	Edge Histogram Dir.		**5**

As detailed in Table 1, some of the basic color (e.g. MPEG-7 Dominant Color Descriptor, HSV color histogram and Color Structure Descriptor [7]), texture (e.g. Gabor [8], Local

Binary Pattern [9], and Ordinal Co-occurrence Matrix [10]) and edge (e.g. Edge Histogram Direction [7]) features, are extracted. Some of them are created with different parameters to extract several sub-features and the total feature dimension is obtained as 2335. Such a high feature space dimension can thus give us opportunity to test the performance of the proposed CNBC framework against the "curse of dimensionality" and *scalability* with the varying number of features.

B. Classification Results

Both databases are partitioned in such a way that the majority (55%) of the items is spared for testing and the rest was used for evolving the CNBC. To demonstrate the feature scalability property of the CNBC, we evolved two CNBCs individually using 7 (FVs 1, 3, 5, 11, 12, 13 and 14 in Table 1 with total dimension of 188) and 14 (all FVs with total dimension of 2335) features. Therefore, the first CNBC has 7+1=8 BCs and the second has 14+1=15 BCs in each NBC. The evolution (and training) parameters are as follows: For MD-PSO, we use the termination criteria as the combination of the maximum number of iterations allowed (*iterNo* = 100) and the cut-off error ($\varepsilon_C = 10^{-4}$). Other parameters were empirically set as: the swarm size, *S*=50, $V_{\max} = x_{\max}/5 = 0.2$ and $VD_{\max} = 5$, respectively. For exhaustive BP, the learning parameter is set as $\lambda = 0.01$ and iteration number is 20. We use the typical activation function: *hyperbolic tangent* ($\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$). For the AS, we used simple configurations with the following range arrays: $R_{\min} = \{N_i, 8, 2\}$ and $R_{\max} = \{N_i, 16, 2\}$, which indicate that besides the single layer perceptron (SLP), all MLPs have only a single hidden layer, i.e. $L_{\max} = 2$, with no more than 16 hidden neurons. Besides the SLP, the hash function enumerates all MLP configurations in the AS, as shown in Table 2. Finally, for both evolution methods, the exhaustive BP and MD PSO, $N_R = 10$ independent runs are performed. Note that for exhaustive BP, this corresponds to 10 runs for each configuration in the AS.

Table 2: The architecture space used for MLPs.

Dim.	*Conf.*	Dim.	*Conf.*	Dim.	*Conf.*
0	N_i x 2	**6**	N_i x6x2	**12**	N_i x12x2
1	N_i x1x2	**7**	N_i x7x2	**13**	N_i x13x2
2	N_i x2x2	**8**	N_i x8x2	**14**	N_i x14x2
3	N_i x3x2	**9**	N_i x9x2	**15**	N_i x15x2
4	N_i x4x2	**10**	N_i x10x2	**16**	N_i x16x2
5	N_i x5x2	**11**	N_i x11x2		

Table 3 presents the classification performances achieved for *Corel_10* database by both evolutionary techniques over the sample AS given in Table 2. The results indicate that MD PSO achieves the lowest MSE and CE levels (and hence the best results) within the training set whereas vice versa is true for the exhaustive BP within the test set. CNBC in general demonstrates a solid robustness against the major feature dimension increase (i.e. from 7 (188-D) to 14 sub-features (2335-D)) since the classification performance does not show any deterioration, on contrary, with both techniques a better performance is achieved with enhanced generalization ability. This is an expected outcome since CNBC can benefit from the additional discrimination capability of each incoming (sub-) feature thanks to its "Divide and Conquer" type design where an efficient feature selection scheme is embedded.

Table 3: Classification performance of each evolution method per feature set for *Corel_10* database.

Feature Set	Evol. Method	Train MSE	Train CE %	Test MSE	Test CE %
7 sub-features	**MDPSO**	**0.32**	**3.55**	1.61	20.18
	BP	0.45	4.88	**1.27**	**18.36**
14 sub-features	**MDPSO**	**0.28**	**3.18**	1.46	18.63
	BP	0.34	3.33	**1.25**	**14.54**

Table 4 presents the confusion matrix of the best classification result over the test set, i.e. achieved by the exhaustive BP method using 14 sub-features. It is worth noting that the major source of error results from the confusion between the 2nd (*Beach*) and 9th (*Mountain*) classes where low-level features cannot really discriminate due to excessive color and texture similarities among those classes. This is also true for the 6th class (*Elephant*) from which the background of some images share a high similarity with both classes.

Table 4: Confusion matrix of the evolution method, which gave the best (lowest) test CE in Table 3.

Actual		*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Truth	**1**	**42**	2	1	1	0	5	0	0	1	3
	2	2	**37**	4	1	0	0	0	1	9	1
	3	2	3	**46**	1	0	1	0	0	1	1
	4	2	0	0	**53**	0	0	0	0	0	0
	5	0	0	0	0	**55**	0	0	0	0	0
	6	2	4	2	0	0	**37**	0	1	8	1
	7	1	0	0	0	0	0	**53**	1	0	0
	8	0	0	0	0	0	0	0	**55**	0	0
	9	0	8	1	0	0	0	0	0	**46**	0
	10	1	1	0	1	1	2	1	0	2	**46**

The CNBC evolutions so far performed are much alike to the (batch) training of traditional classifiers (such as ANNs, k-NN, Bayesian) where the training data (the features) and (number of) classes are all fixed and the entire GTD is used during the training (evolution). As detailed earlier, the CNBC can also be evolved incrementally, i.e. incremental evolutions can be performed whenever new features/classes can be introduced and the CNBC can dynamically create new BCs and/or NBCs as the need arises. In order to evaluate the incremental evolution performance, the training dataset is divided into three distinct partitions, each of which contains 5 (classes 1-5), 3 (classes 6-8) and 2 (classes 9 and 10) classes, respectively. Therefore, three stages of incremental evolutions have been performed where at each stage the CNBC is further evolved only with the particular dataset partition, which belongs to the new classes in that partition. After the first phase,

only three out of five existing NBCs were incrementally evolved over the training dataset of the three new classes (classes 6-8). Similarly, at the third phase, three out of 8 NBCs did not undergo for incremental evolution since their classification accuracy over the training dataset of those new classes (9 and 10) are already above the minimum classification accuracy threshold (95%) required.

Due to space limitations, we had to skip details on the classification performances achieved at the intermediate stages. Table 5 presents the final classification performance of each evolution method per feature set. The results indicate a few percent losses on both training and test classification accuracies, which can be expected since the incremental evolution was purposefully skipped for some NBCs whenever they surpass 95% classification accuracy over the training dataset of the new (emerging) classes. This means, for instance, some NBCs (e.g. the one corresponds to class 4, the *Bus*) evolved with only over a subset of the entire training dataset.

Table 5: Final classification performance of 3-stage incremental evolution per evolution method and feature set for *Corel_10* database.

Feature Set	Evol. Method	Train MSE	Train CE %	Test MSE	Test CE %
7 sub-features	MDPSO	1.36	6.89	2.61	28.63
	BP	**0.82**	**4.22**	**1.85**	**21.63**
14 sub-features	MDPSO	1.23	**6.66**	2.39	26.36
	BP	**0.91**	7.55	**1.83**	**23.81**

Finally, the CNBC evolution for *Corel_Caltech_30* database allows testing and evaluation of its classification performance when the database size and number of classes are significantly increased. For both evolution techniques, we used the same parameters as presented earlier except that the number of epochs (iterations) for BP and MD PSO were increased to 200 and 500 in order to compensate the increase in the database size. Table 6 presents the classification performances of each evolution method per feature set. Due to space limitations we had to skip the results from incremental evolutions in this dataset. As compared with the results from *Corel_10* database in Table 3, it is evident that both evolution methods achieved a similar classification performance in the training set (i.e. similar train CEs) whilst certain degradation occurs in the classification accuracy in test set (i.e. 10-15% increase in the test CEs). This is an expected outcome since the lack of discrimination within those low-level features can eventually yield a poorer generalization especially when the number of classes is tripled.

Table 6: Classification performance of each evolution method per feature set for *Corel_Caltech_30* database.

Feature Set	Evol. Method	Train MSE	Train CE	Test MSE	Test CE
7 sub-features	MD PSO	0.54	8.1	2.3	**33.40**
	BP	**0.24**	**2.95**	**2.16**	34.67
14 sub-features	MD PSO	0.33	5.47	**2.52**	36.33
	BP	**0.074**	**1.31**	2.69	**33.86**

Figure 5: 4x2 sample queries in *Corel_10* (qA and qB), and *Corel_Caltech_30* (qC and qD) databases Top-left is the query image.

C. Retrieval Results

The traditional retrieval process in MUVIS is based on the query by example (QBE) operation. The (sub-) features of the query item are used for (dis-) similarity measurement among all the features of the visual items in the database. Ranking the database items according to their similarity distances yields the retrieval result. The traditional (dis-) similarity measurement in MUVIS is accomplished by applying a distance metric such as L2 (*Euclidean*) between the feature vectors of the query and each database item. When a CNBC is used for the purpose of retrieval, the same (L2) distance metric is now applied to the class vectors at the output layer of the CNBC (10x2=20-D for *Corel_10* and 30x2=60-D for *Corel_Caltech_30* databases). In order to evaluate the retrieval performances with and without CNBC, we use average precision (*AP*) and average normalized modified retrieval rank (ANMRR) measures, both of which are computed querying *all* images in the database (i.e. batch query) and within a retrieval window equal to the number of ground truth images, $N(q)$ for each query q. This henceforth makes the *AP* identical to average recall and average F1 measures, too.

Over each database, four batch queries are performed to compute the average retrieval performances, two with and two without using the CNBC. Whenever used, the CNBC is evolved with the MD PSO and the exhaustive BP, the former with 7 and the latter with 14 sub-features, respectively. As listed in Table 7, it is evident that the CNBC can significantly enhance the retrieval performance regardless of the evolution method, the feature set and the database size. The results (without CNBC) in the table also confirm the enhanced discrimination obtained from the larger feature set, which led to better classification performance and in turn, leads to a better retrieval performance.

Table 7: Retrieval performances (%) of the four batch queries in each MUVIS databases.

Feature Set	Evol. Method	*Corel_10*		*Corel_Caltech_30*	
		ANMRR	AP	ANMRR	AP
7 sub-features	MD PSO	**33.09**	**64.01**	**43.04**	**54.47**
	None	55.81	42.15	60.21	37.80
14 sub-features	BP	**22.21**	**76.20**	**32.00**	**65.37**
	None	47.19	50.38	62.94	34.92

For visual evaluation, Figure 5 presents four typical retrieval results with and without using the proposed CNBC framework. All query images are selected among the test set and the query is processed within the entire database.

V. Conclusions

In this paper, a novel CNBC framework is introduced to address the problem of efficient and accurate content-based classification and retrieval within large image databases. CNBC is a *Divide and Conquer* type of approach, which reduces both feature and class vector dimensions for individual classifiers significantly to enable the use of as compact classifiers as possible. Such compact classifiers can be evolved and trained better than a single yet more complex classifier. The optimum classifier for each classification problem at hand can be searched separately and at a given time, this allows to create new dedicated classifiers (BCs) for discriminating a certain class type from the others with the use of a single (sub-) feature to accommodate new features or to create a new NBC to allow introduction of new classes. Each (incremental) evolution session "learns" from the current best classifier and can improve it further, possibly using another configuration in the AS. Moreover, when trained properly, the fuser BC can correct the erroneous classification of any BC in the input layer, which further increases the classification accuracy. Thus classification performance, the main advantages of the proposed framework are the improved classification performance and the efficient solution provided to the problems of *scalability* and *dynamic adaptability* by allowing both feature space dimensions and the number of classes in a database to be unlimited and dynamic (incremental). Furthermore, the CNBC framework is designed for both online (incremental) and offline (batch) evolutions, which can be performed in multiple runs. During each run, any new configuration can replace the current one in the AS if it outperforms it. Such an evolutionary update mechanism ensures that the AS containing the best configurations, is always kept intact and that only the best configuration at any given time is used for classification and retrieval.

Although the results indicate that all the aforementioned objectives have been successfully fulfilled, even higher accuracy levels can still be expected from the CNBC framework with the addition of new powerful features with superior discrimination and content description capabilities.

References

[1] S. Kiranyaz, T. Ince, A. Yildirim and M. Gabbouj, "Evolutionary Artificial Neural Networks by Multi-Dimensional Particle Swarm Optimization", Neural Networks, vol. 22, pp. 1448 – 1462, doi:10.1016/j.neunet.2009.05.013, Dec. 2009.

[2] S. Kiranyaz, T. Ince, A. Yildirim and M. Gabbouj, "Fractional Particle Swarm Optimization in Multi-Dimensional Search Space", *IEEE Trans. on Systems, Man, and Cybernetics – Part B*, pp. 298 – 319, vol. 40, No. 2, 2010.

[3] Y. Chauvin and D. E. Rumelhart, "Back Propagation: Theory, Architectures, and Applications," Lawrence Erlbaum Associates Publishers, UK, 1995.

[4] MUVIS. http://muvis.cs.tut.fi/

[5] Corel Collection / Photo CD Collection (www.corel.com)

[6] L. Fei-Fei, R. Fergus and P. Perona, "Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories", *IEEE CVPR Workshop on Generative-Model Based Vision*, vol. 12, pp.178, 2004.

[7] B. S. Manjunath, J.-R. Ohm, V. V. Vasudevan, and A. Yamada, "Color and Texture Descriptors", *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 11, pp. 703-715, Jun. 2001.

[8] B. Manjunath, P. Wu, S. Newsam, H. Shin, "A texture descriptor for browsing and similarity retrieval", Journal of Signal Processing: Image Communication, vol. 16, pp. 33-43, Sep. 2000.

[9] T. Ojala, M. Pietikainen, D. Harwood, "A comparative study of texture measures with classification based on feature distributions", Pattern Recognition, vol. 29, pp. 51-59, 1996.

[10] M. Partio, B. Cramariuc, M. Gabbouj, "An Ordinal Co-occurrence Matrix Framework for Texture Retrieval", EURASIP Journal on Image and Video Processing, vol. 2007, Article ID 17358, 15 pages, 2007. doi:10.1155/2007/17358.

Publication III

Serkan Kiranyaz, Jenni Pulkkinen, Turker Ince, Moncef Gabbouj, "Multi-dimensional Evolutionary Feature Synthesis for Content-based Image Retrieval," *IEEE 18th International Conference on Image Processing (ICIP 2011)*, pp. 3645 – 3648, Sep. 2011.

MULTI-DIMENSIONAL EVOLUTIONARY FEATURE SYNTHESIS FOR CONTENT-BASED IMAGE RETRIEVAL

Serkan Kiranyaz, Jenni Pulkkinen, Turker Ince and Moncef Gabbouj*
Dept. of Signal Processing, Tampere University of Technology, Finland, {firstname.lastname}@tut.fi
* Faculty of Computer Science, Izmir University of Economics, Izmir, Turkey, turker.ince@ieu.edu.tr

ABSTRACT

Low-level features (also called as *descriptors*) play a central role in content-based image retrieval (CBIR) systems. Features are various types of information extracted from the content and represent some of its characteristics or signatures. However, especially the (low-level) features, which can be extracted automatically usually lack the discrimination power needed for accurate description of the image content and may lead to a poor retrieval performance. In order to efficiently address this problem, in this paper we propose a multi-dimensional evolutionary feature synthesis technique, which seeks for the optimal linear and non-linear operators so as to synthesize highly discriminative set of features in an optimal dimension. The optimality *therein* is sought by the multi-dimensional particle swarm optimization method along with the fractional global-best formation technique. Clustering and CBIR experiments where the proposed feature synthesizer is evolved using only the minority of the database, demonstrate a significant performance improvement and exhibit a major discrimination between the features of different classes.

Index Terms— Evolutionary feature synthesis, Content-based image retrieval, multi-dimensional particle swarm optimization.

1. INTRODUCTION

It can be foreseen that the future CBIR systems require a decisive solution for the well-known "Semantic Gap" problem. In general, narrowing the semantic gap basically requires advanced approaches that depend on a central element to describe the image content: the *features*. Features are the fundamental elements of CBIR. They are the information extracted from an image, represented in a suitable way, stored in an index, and used during query processing. They characterize the content characteristics and signatures. However, especially the (low-level) features, which can be extracted automatically, usually lack the discrimination power needed for accurate retrievals in large image collections. Furthermore, among a vast number of feature extraction techniques, the question of how to select the most appropriate set of features still remained unanswered.

The efforts addressing the aforementioned problems can be categorized into two feature transformation types: feature selection and feature synthesis. The former does not change the original features; instead selects a particular subset of them to be used in CBIR. So no matter how efficient the feature selection method may be, the final outcome is nothing but a subset of the original features and may still lack the discrimination power needed for an efficient CBIR. The latter basically performs a transformation, linear and/or non-linear, to synthesize new features. For both approaches evolutionary algorithms (EAs) [1] such as Genetic Algorithm (GA) [2] and Genetic Programming (GP) [3] are mainly used.

Evolutionary feature synthesis (EFS) is still in its infancy as there are only few successful methods proposed up to date. In a recent work [4], GA was used to evolve texture features for CBIR on skin lesions. Although only 6 simple arithmetic operators were used in a small-scale database with 100 images, the retrieval performance (i.e. the precision) was slightly improved (7%) with the proper settings of parameters. In [5], co-evolutionary GP (CGP) was utilized in sub-populations to synthesize features for object recognition. Although some performance improvement has been observed, in both methods, the improvement (in retrieval performance or in recognition rate) may be insignificant or as in [5], original features may even surpass the synthesized features in some cases. This is due to several facts. First of all, only few (non-)linear operators are used in order to avoid a high search space dimension due to the fact that the probability of getting trapped into a local optimum significantly increases in higher dimensions [6]. Similarly in both methods, the synthesized feature space dimension is kept quite low not to increase computational complexity, i.e. in [5] the number of sub-populations is equal to the dimension of the synthesized (or the so-called composite) features. Furthermore, both methods suffer from the manual and sub-optimal setting of several GA and CGP parameters (e.g. in [5] there are more than 10 parameters that should be properly set in advance). The most critical drawback among all is that both GA and CGP can only work in a search space with a dimension fixed *a priori*. This leads that the optimum dimension for the feature synthesis will remain unknown.

To address these problems, in this paper we propose an evolutionary feature synthesis technique that is based on multi-dimensional particle swarm optimization (MD-PSO) [7], which can find the optimum dimension of the solution space and hence voids the need of fixing the dimension of the solution space in advance. MD-PSO can also work along with the fractional global best formation scheme (FGBF) [7] to avoid the premature convergence problem. With the proper encoding scheme that encapsulates several linear and non-linear operators (applied to a set of selected features), and their scaling factors (weights), MD-PSO particles can therefore, perform an evolutionary search to find out the optimal feature synthesizer to generate new features in the optimal dimension. The optimality *therein* can be set by such a proper fitness measure that maximizes the overall CBIR (or clustering) performance.

The rest of the paper is organized as follows. The motivation and details of the proposed EFS along with the underlying evolutionary search technique, the MD-PSO, are presented in Section 2. Section 3 introduces EFS experiments that are performed in *Corel* image databases, and presents the clustering and retrieval results along with the comparative evaluations. Finally, Section 4 concludes the paper and discusses topics for future work.

2. EVOLUTIONARY FEATURE SYNTHESIS

2.1. The Motivation

As mentioned earlier, the motivation behind the proposed evolutionary feature synthesis (EFS) technique is to maximize the discrimination power of low-level features so that CBIR performance can be significantly improved. In a broader sense, well-known classifiers such as Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) can be thought as a special kind of feature synthesizers. Commonly ANNs used as classifiers take the original feature vector as an input and, in an optimal case, their output is a vector corresponding to the image class (e.g. for c=1, {1, 0, … , 0}). Thus the ANNs try to learn a feature synthesizer that transforms each feature vector in a certain class to one corner of the d-dimensional cube (where d is the number of classes). SVMs, on the other hand, attempt to transform the original features into a new

(higher) dimension where linear separation is possible. A major drawback of these kinds of feature synthesizers is the critical choice of the (non-)linear kernel (or activation) function that may not be a proper choice for the problem in hand. Consider for instance, two sample feature synthesizers (*FS-1* and *FS-2*) illustrated in Figure 1 where for illustration purposes features are only shown in 1-D and 2-D, and only two-class problem is considered. In the case of *FS-1*, SVM with a polynomial kernel in quadratic form can make the proper transformation into 3-D so that the new (synthesized) features are linearly separable. However, for *FS-2*, a sinusoid with a proper frequency, *f,* should *instead* be used for a better class discrimination. Therefore, searching for the right transformation (and hence for the linear and non-linear operators within) is of paramount importance, which is not possible for static (or fixed) ANN and SVM configurations. Since there is no feature selection (all features are used to synthesize a single synthesized feature output), during training especially multi-layer ANNs may further suffer from the high complexity in terms of massive number of parameters (weights and thresholds). They are also prone to limited performance due to the sub-optimal dimension setting for the synthesized features. In order to maximize the discrimination among classes, the dimension into which new features are synthesized, should also be optimized by the evolutionary search technique.

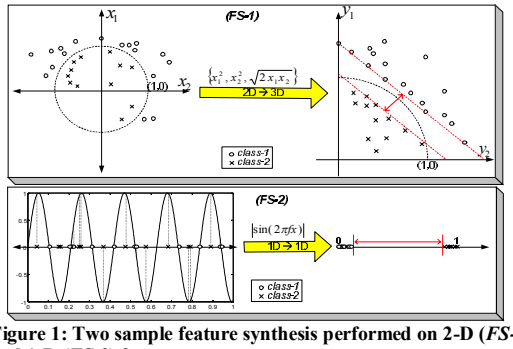


Figure 1: Two sample feature synthesis performed on 2-D (*FS-1*) and 1-D (*FS-2*) feature spaces.

2.2. Multi-Dimensional Particle Swarm Optimization

As the evolutionary search method, we use the multi-dimensional (MD) extension of the basic PSO (*bPSO*) method, the so-called MD-PSO, recently proposed in [7]. Instead of operating at a fixed dimension *d*, the MD-PSO algorithm is designed to seek both positional and dimensional optima within a given dimension range, $\{D_{\min}, D_{\max}\}$. In order to accomplish this, each particle has two sets of components, each of which has been subjected to one of the two independent and consecutive processes. The first one is a regular positional PSO, i.e. the traditional velocity updates and due positional shifts in *N* dimensional search (solution) space. The second one is a dimensional PSO, which allows the particle to navigate through dimensions. Accordingly, each particle keeps track of its last position, velocity and personal best position (*pbest*) in a particular dimension so that when it re-visits the same dimension at a later time, it can perform its regular "positional" update using this information. The dimensional PSO process of each particle may then move the particle to another dimension where it will remember its positional status and will be updated within the positional PSO process at this dimension, and so on. The swarm, on the other hand, keeps track of the *gbest* particle in each dimension, indicating the best (global) position so far achieved. Similarly, the dimensional PSO process of each particle uses its personal best dimension in which the personal best fitness score has so far been achieved. Finally, the swarm keeps track of the global best dimension, *dbest*, among all the personal best dimensions. The *gbest* particle in the *dbest* dimension represents the optimum solution and dimension, respectively.

In a MD-PSO process at time (iteration) *t*, each particle *a* in the swarm with *S* particles, $\xi = \{x_1,..,x_a,..,x_S\}$, is represented by the following characteristics:

$xx_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the position of particle *a*, in dimension $xd_a(t)$

$vx_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the velocity of particle *a*, in dimension $xd_a(t)$

$xy_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the personal best position of particle *a*, in dimension $xd_a(t)$

$gbest(d)$: Global best particle index in dimension *d*

$x\hat{y}_j^d(t)$: j^{th} component (dimension) of the global best position of swarm, in dimension *d*

$xd_a(t)$**:** Dimension of particle *a*

$vd_a(t)$**:** Dimensional velocity of particle *a*

$x\tilde{d}_a(t)$**:** Personal best dimension component of particle *a*

Further algorithmic details of the MD-PSO method can be found in [7] and are skipped in this paper due to the page limitations.

2.3. Evolutionary Feature Synthesis by MD-PSO

2.3.1. The Overview

As shown in Figure 2, the proposed evolutionary feature synthesis (EFS) can be performed in one or several runs where each run can further synthesize the features generated from the previous run. The number of runs, *R*, can be specified in advance or adaptively determined, i.e. runs are carried out until a point where the fitness improvement is no longer significant. The EFS dataset can be the entire image database or a certain sub-section of it where the ground truth is available. If there is more than one Feature eXtraction (FeX) module, an individual feature synthesizer can be evolved for each module and once completed, each set of features extracted by an individual FeX module can then be passed through the individual synthesizers to generate new features for CBIR.

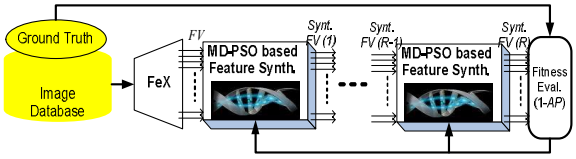


Figure 2: The block diagram of Feature eXtraction (FeX) and the proposed EFS technique with *R* runs.

2.3.2. Encoding of the MD-PSO particles

The position of each MD-PSO particle in a dimension $d \in [D_{\min}, D_{\max}]$, represents a potential feature synthesizer, which generates a *d*-dimensional feature vector using a set of applied operators over some selected features within *N*-dimensional source (original) feature vector. Therefore, the j^{th} component (dimension) of the position of particle *a*, in dimension *d* and at an iteration *t*,

$xx_{a,j}^{d}(t), j \in [0, d-1]$, synthesizes the j^{th} feature of the d-dimensional feature vector.

Table 1: *F*=20 operators used in evolutionary synthesis.

θ_i	Formula	θ_i	Formula	θ_i	Formula
0	*A*	**7**	*A-B*	**14**	*tan(2πfAB)*
1	*B*	**8**	*AB*	**15**	*tan(2πf (A+B))*
2	*max(A, B)*	**9**	*A/B*	**16**	$0.5\exp(-(A+B)^2)$
3	*min(A, B)*	**10**	*sin(2πfAB)*	**17**	$0.5\exp(-(A-B)^2)$
4	A^2	**11**	*cos(2πfAB)*	**18**	$\log((A+B)^2)$
5	B^2	**12**	*sin(2πf (A+B))*	**19**	$\log((A-B)^2)$
6	*A+B*	**13**	*cos(2πf(A+B))*		

Along with the operators and the feature selection, encoding of the MD-PSO particles is designed to enable a feature *scaling* mechanism with the proper weights. As illustrated in Figure 3, $xx_{a,j}^{d}(t), j \in [0, d-1]$ is a $2K+1$ dimensional vector encapsulating 1) the *selection* of $K+1$ (input) features (with feature indices, $\alpha 1, \beta 1, ..., \beta K \in [0, N-1]$), 2) their weights ($0 \le w_{\alpha 1}, w_{\beta i} < 1$) and 3) K operators (via indices, $\theta_{i \in [1,K]}$ enumerated in Table 1). As a result of this K-depth feature synthesis, the j^{th} element of the d-dimensional feature vector can be generated as,

$$y_j = \Theta_K\left(w_{\beta K} x_{\beta K}, \Theta_{K-1}\left(..., \Theta_2\left(w_{\beta 2} x_{\beta 2}, \Theta_1\left(w_{\beta 1} x_{\beta 1}, w_{\alpha 1} x_{\alpha 1}\right)\right)...\right)\right) \quad (1)$$

$$xx_{a,j}^{d} = \begin{bmatrix} \mathrm{A}_1 \\ \mathrm{B}_1 \\ \theta_1 \\ \mathrm{B}_2 \\ \theta_2 \\ \cdots \\ \cdots \\ \mathrm{B}_K \\ \theta_K \end{bmatrix}$$

where,

$j \in [0, d-1]$, $\mathrm{A}_1, \mathrm{B}_i \in \Re[0, N)$, $\theta_i \in [1, F]$, $i \in [1, K]$

Let:

$\alpha 1 = \lfloor \mathrm{A}_1 \rfloor \in [0, N-1]$, $\beta i = \lfloor B_i \rfloor \in [0, N-1]$

$w_{\alpha 1} = \mathrm{A}_1 - \alpha 1$ *and* $w_{\beta i} = \mathrm{B}_i - \beta i, 0 \le w_{\alpha 1}, w_{\beta i} < 1$

$\Theta_i \equiv Operator\ (\theta_i)$

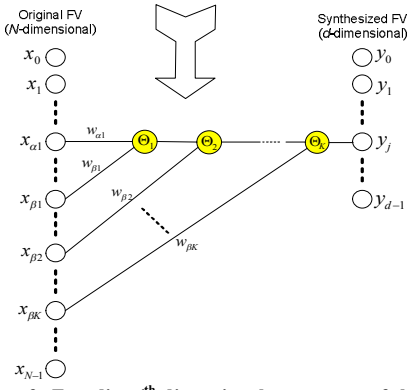


Figure 3: Encoding j^{th} dimensional component of the particle *a* in dimension *d* for *K*-depth feature synthesis.

Note that letting $K=N+1$ and fixing the operator Θ_K to a typical activation function such as *sigmoid* or *tangent hyperbolic*, and the rest, $\Theta_{i \in [1, K-1]}$, to "+" operator (*Operator*(6) in Table 1) makes the proposed feature synthesis technique equivalent to a basic feed-forward ANN (or single-layer perceptron, SLP). Similarly, if more than one EFS runs performed ($R>1$), the overall scheme is equivalent of a typical MLP. In short, feed-forward ANNs are indeed a special case of the proposed EFS technique, yet the most complex one due to the usage of *all* input features ($K=N+1$), which voids the feature selection. Moreover, this makes it the most limited case, since it uses only two operators among many possibilities. Therefore, the focus is drawn to achieve a low complexity by selecting only a reasonable number of features (with a low K value) and performing as few MD-PSO runs as necessary (with a low R value).

2.3.3. The Fitness Function

Since the main objective is to maximize the CBIR performance, a straightforward fitness function (to be *minimized*) is the inverse average precision (*-AP* or *1-AP*) or alternatively, the average normalized modified retrieval rank (*ANMRR*), both of which can directly be computed by querying all images in the ground truth dataset and averaging individual precision or *NMRR* scores. This, however, may turn to a costly operation especially for large databases with many classes. An alternative fitness function that seeks to maximize discrimination among distinct classes can be a clustering validity index (CVI) where each cluster corresponds to a distinct class in the database. CVI can be formed with respect to two widely used criteria in clustering: , intra-cluster compactness and inter-cluster separation.

For each potential EFS encoded in a MD-PSO particle, the CVI computed over the d-dimensional synthesized features, $Z = \{z_p, z_p \in c_j\}$, for each class, c_i, $i \in [1, C]$, can be computed as in Eq. (2).

$$f(\mu_{c,\Sigma}, Z) = FP\ (\mu_{c,\Sigma}, Z) + \left(Q_e(\mu_{c,\Sigma}, Z) / d_{\min}(\mu_{c,\Sigma})\right)$$

$$where \quad Q_e(\mu_{c,\Sigma}, Z) = \frac{1}{C} \sum_{i=1}^{C} \frac{\sum_{\forall z_p \in c_i} \left\| \mu_{c,i} - z_p \right\|}{\left\| c_i \right\|} \quad (2)$$

where $\mu_{c,\Sigma}$ be the centroid vector computed for all classes, and $Q_e(\mu_{c,\Sigma}, Z)$ is the quantization error (or the average intra-cluster distance). $d_{\min}$ is the minimum centroid (inter-cluster) distance among all classes and $FP(\mu_{c,\Sigma}, Z)$ is the number of *false positives* i.e. synthesized feature vectors which are closer to another class centroid than their own. So the minimization of the validity index $f(\mu_{c,\Sigma}, Z)$ will simultaneously try to minimize the intra-cluster distances (for better *Compactness*) and maximize the inter-cluster distance (for better *Separation*), both of which lead to a low $FP(\mu_{c,\Sigma}, Z)$ value or in the ideal case $FP(\mu_{c,\Sigma}, Z) = 0$, meaning that each synthesized feature is in the closest proximity of its own class centroid, thus leading to high discrimination.

3. EXPERIMENTAL RESULTS

For the EFS experiments performed in this section, we used MUVIS framework [8], to create and to index two image databases from *Corel* image collection [9]: 1) *Corel_10* Image Database: There are 1000 medium resolution (384x256 pixels) images obtained from Corel repository [9] covering 10 diverse classes: 1 - *Natives*, 2 - *Beach*, 3 - *Architecture*, 4 - *Bus*, 5 - *Dino Art*, 6 - *Elephant*, 7 - *Flower*, 8 - *Horse*, 9 - *Mountain*, and 10 – *Food*, and 2) *Corel_5* Image Database: Composed of images taken from the first 5 diverse classes in *Corel_10* database. In order to evaluate the proposed EFS technique and test its efficacy for both CBIR and clustering, we purposefully extract a well-known low-level descriptor, 64-bins RGB color histogram (unit normalized), which has a limited discrimination power and a severe deficiency for a proper content

description. The performance of the proposed EFS technique for improving CBIR accuracy is tested over *Corel_5* database by using the (inverse) average precision (1-*AP*) directly as the fitness function. Similarly, assigning the CVI given in Eq. (2) as the fitness function, the clustering performance of the features synthesized is tested over *Corel_10* database. Both databases are partitioned in such a way that the majority (55%) of the images is spared for testing and the rest (45%) is used as the EFS dataset. For MD-PSO, we use the swarm size, S=100 and the termination criteria as the maximum number of iterations allowed (*iterNo*=1000). The dimensional range, $[D_{\min}, D_{\max}]$, that the optimum dimension for EFS will be searched is set as, $[40, 120]$. The rest of the internal MD-PSO parameters are used as recommended in [7].

The depth of the EFS, K, is set as low as 3 for CBIR in *Corel_5* to test its performance for such a quite low value and also to avoid further complexity. For experiments in *Corel_10, K* is otherwise set to 10, which allows selecting (up to) 10 features among 64 (RGB histogram bins). Finally, multiple runs are allowed as long as a significant *AP* improvement between consecutive runs is observed (i.e. > 5%).

Table 2: Fitness scores (1-*AP*) and CBIR performances of the original and synthesized features per run.

	Original (64-D)	Run-1 (75-D)	Run-2 (56-D)	Run-3 (91-D)
Fit. Score	0.429	0.337	0.211	**0.146**
***AP* (%)**	51.1	59.01	67.1	**70.48**
***ANMRR* (%)**	47.2	39.2	30.72	**27.67**

Table 2 presents the fitness scores (1-*AP*) achieved by the original and synthesized features in the best dimension converged per MD-PSO run over the EFS dataset of the *Corel_5* database. Furthermore, the overall CBIR performances computed by querying all databases items (batch query) are also presented with standard *AP* and *ANMRR* measures.

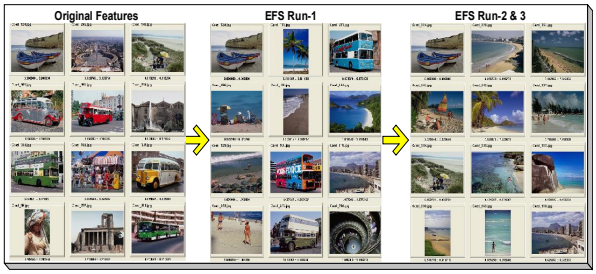


Figure 4: Four sample queries from class 2 (*Beach*) performed using original and synthesized features at each run. Top-left is the query image.

Figure 4 shows four sample queries, each of which is performed either using the original features or synthesized features per EFS run. The query operation using the synthesized features from the last two EFS runs (2 and 3) retrieved the same 12 images. It is obvious that the synthesized features further improve their description power at each run and in turn, better CBIR performance is achieved. Finally, Table 3 presents the clustering performances of the original and synthesized features measured by the CVI ($f(\mu_{c,\Sigma}, Z)$) given in Eq. (2) and also by the number of false positives ($FP(\mu_{c,\Sigma}, Z)$). As discussed earlier, this is an alternative way of evaluating the improvement on the discrimination power of the synthesized features from which it is evident that a significantly higher clustering performance can be achieved.

Table 3: Fitness score (CVI) and the number of false positives (*FP*) of the original and synthesized features.

Dataset:	**EFS**		***Corel_10***	
Features:	**Original**	**Synth.**	**Original**	**Synth.**
Fit. Score (CVI)	186.98	**69.28**	431.17	**268.65**
FP	144	**46**	357	**236**

4. CONCLUSIONS

In this paper, we proposed a multi-dimensional evolutionary feature synthesis technique, which aimed to improve the discrimination among the low-level features and in turn to enhance the CBIR and clustering performances. As the evolutionary search technique, we used MD-PSO and FGBF that have recently been proposed as a cure to common drawbacks of the PSO family. Particularly, MD-PSO based EFS has the advantage to search the optimal dimension for the synthesized feature vectors. This is, to our knowledge, an unprecedented advantage that none of the earlier feature synthesis methods have accomplished before. Moreover, by means of the proposed encoding technique, MD-PSO particles can perform an evolutionary search for the optimum operators, scales (weights) and selection of the (original) features, all simultaneously in an interleaved way. As discussed earlier, this alone provides a higher flexibility and better feature (or data) adaptation than the regular ANNs and SVM classifiers, since the proposed EFS technique can utilize a large set of (non-linear) operators and have the advantage of selecting proper features. Finally, the proposed technique does not have critical parameters or thresholds that may significantly affect the performance.

Experimental results over benchmark image databases have demonstrated that the proposed EFS technique has synthesized more discriminative features with a lower CVI and with a significantly higher CBIR performance. We can conclude that as long as some ground truth is available for an image database (i.e. via relevance feedback), the low-level feature extraction will just be the initial step, because features with a higher discrimination (and description) capability can effectively be synthesized from them.

5. REFERENCES

[1] T. Back and H.P. Schwefel, "An overview of evolutionary algorithm for parameter optimization", *Evolution. Comput. 1*, pp. 1–23, 1993.

[2] D. Goldberg, Genetic *Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, pp. 1-25. MA, 1989.

[3] J. Koza, *Genetic Programming: On the Programming of Computers by means of Natural Selection,* MIT Press, Cambridge, 1992.

[4] L. Ballerini, X. Li, R. B. Fisher, Be. Aldridge, J. Rees, "Content-Based Image Retrieval of Skin Lesions by Evolutionary Feature Synthesis," EvoApplications, Istanbul, Turkey, pp.312-319, 2010.

[5] Y. Lin, and B. Bhanu, "Evolutionary Feature Synthesis for Object Recognition", IEEE Trans. on Man and Cybernetics – Part C, vol. 35, No. 2, pp. 156-171, May 2005.

[6] F. Van den Bergh, "An Analysis of Particle Swarm Optimizers", PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[7] S. Kiranyaz, T. Ince, A. Yildirim and M. Gabbouj, "Fractional Particle Swarm Optimization in Multi-Dimensional Search Space", *IEEE Trans. on Systems, Man, and Cybernetics – Part B*, pp. 298 – 319, vol. 40, No. 2, 2010.

[8] MUVIS. http://muvis.cs.tut.fi/

[9] Corel Collection / Photo CD Collection (www.corel.com)

Publication IV

Serkan Kiranyaz, Jenni Pulkkinen, Moncef Gabbouj, "Multi-dimensional Particle Swarm Optimization in Dynamic Environments," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2212 – 2223, Mar. 2011.

Expert Systems with Applications 38 (2011) 2212–2223

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Multi-dimensional particle swarm optimization in dynamic environments

Serkan Kiranyaz *, Jenni Pulkkinen, Moncef Gabbouj [1]

Department of Signal Processing, Tampere University of Technology, P.O. Box 553 FIN-33101, Tampere, Finland

ARTICLE INFO

Keywords:
Particle swarm optimization
Multi-dimensional search
Fractional Global Best Formation

ABSTRACT

Particle swarm optimization (PSO) was proposed as an optimization technique for static environments; however, many real problems are dynamic, meaning that the environment and the characteristics of the global optimum can change in time. In this paper, we adapt recent techniques, which successfully address several major problems of PSO and exhibit a significant performance over multi-modal and non-stationary environments. In order to address the pre-mature convergence problem and improve the rate of PSO's convergence to the global optimum, Fractional Global Best Formation (FGBF) technique is used. FGBF basically collects all the best dimensional components and fractionally creates an artificial Global Best particle (*aGB*) that has the potential to be a better "guide" than the PSO's native gbest particle. To establish follow-up of local optima, we then introduce a novel multi-swarm algorithm, which enables each swarm to converge to a different optimum and use FGBF technique distinctively. Finally for the multi-dimensional dynamic environments where the optimum dimension also changes in time, we utilize a recent PSO technique, the multi-dimensional (MD) PSO, which re-forms the native structure of the swarm particles in such a way that they can make inter-dimensional passes with a dedicated dimensional PSO process. Therefore, in a multi-dimensional search space where the optimum dimension is unknown, swarm particles can seek for both positional and dimensional optima. This eventually pushes the frontier of the optimization problems in dynamic environments towards a global search in a multi-dimensional space, where there exists a multi-modal problem possibly in each dimension. We investigated both standalone and mutual applications of the proposed methods over the moving peaks benchmark (MPB), which originally simulates a dynamic environment in a unique (fixed) dimension. MPB is appropriately extended to accomplish the simulation of a multi-dimensional dynamic system, which contains dynamic environments active in several dimensions. An extensive set of experiments show that in traditional MPB application domain, FGBF technique applied with multi-swarms exhibits an impressive speed gain and tracks the global peak with the minimum error so far achieved with respect to the other competitive PSO-based methods. When applied over the extended MPB, MD PSO with FGBF can find optimum dimension and provide the (near-) optimal solution in this dimension.

1. Introduction

Many real-world problems are dynamic and thus require systematic re-optimizations due to system and/or environmental changes. Even though it is possible to handle such dynamic problems as a series of individual processes via restarting the optimization algorithm after each change, this may lead to a significant loss of useful information, especially when the change is not too drastic. Since most of such problems have a multi-modal nature, which further complicates the dynamic optimization problem, the need for powerful and efficient optimization techniques is imminent. In the last decade the efforts have been focused on evolutionary algorithms (EAs) (Bäck & Schwefel, 1993) such as Genetic Algorithms (GA) (Goldberg, 1989), Genetic Programming (GP) (Koza, 1992), Evolution Strategies (ES), (Bäck & Kursawe, 1995) and Evolutionary Programming (EP) (Fayyad, Shapire, Smyth, & Uthurusamy, 1996). The common point of all EAs, which have population based nature, is that they may also avoid being trapped in local optima. Thus they can find the optimum solutions; however, this is never guaranteed.

Conceptually speaking, particle swarm optimization (PSO) (Engelbrecht, 2005; Kennedy & Eberhart, 1995; Omran, Salman, & Engelbrecht, 2006), which has obvious ties with the EA family, lies somewhere in between GA and EP. Yet unlike GA, PSO has no complicated evolutionary operators such as *crossover*, *selection* and *mutation* and it is highly dependent on stochastic processes. PSO is originated from the computer simulation of individuals (particles or living organisms) in a bird flock or fish school (Wilson,

* Corresponding author. Tel.: +358 50 4324123; fax: +358 (0)3 3115 4989.
E-mail addresses: serkan.kiranyaz@tut.fi (S. Kiranyaz), jenni.pulkkinen@tut.fi (J. Pulkkinen), moncef.gabbouj@tut.fi (M. Gabbouj).

[1] This work was supported by the Academy of Finland, project No. 213462 (Finnish Centre of Excellence Program (2006–2011).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.08.009

1975), which basically show a natural behavior when they search for some target (e.g. food). Their goal is, therefore, to converge to the global optimum of a possibly non-linear function or system. Similarly, in a PSO process, a swarm of particles (or agents), each of which represents a potential solution to an optimization problem, navigate through the search space. The particles are initially distributed randomly over the search space with a random velocity and the goal is to converge to the global optimum of a function or a system. Each particle keeps track of its position in the search space and its best solution so far achieved. This is the personal best value (the so-called *pbest* in Kennedy & Eberhart (1995)) and the PSO process also keeps track of the global best solution so far achieved by the swarm by remembering the index of the best particle (the so-called *gbest* in Kennedy & Eberhart (1995)). During their journey with discrete time iterations, the velocity of each agent in the next iteration is affected by the best position of the swarm (the best position of the particle *gbest* as the *social* component), the best personal position of the particle (*pbest* as the *cognitive* component), and its current velocity (the *memory* term). Both *social* and *cognitive* components contribute randomly to the velocity of the agent in the next iteration.

There are some efforts for simulating dynamic environments in a standard and configurable way. Some early works such as (Angeline, 1997, 1998; Eberhart & Shi, 2001) use experimental setup introduced by Angeline (1997). In this setup the minimum of the three-dimensional parabolic function, $f(x,y,z) = x^2 + y^2 + z^2$, is moved along a linear or circular trajectory or randomly. Three different update frequencies (200, 1000 and 2000 evaluations) and change severities (0.01, 0.1, 0.5) are used. However, this setup enables testing only in a uni-modal environment. Branke (2008) has provided a publicly available Moving Peaks Benchmark (MPB) to enable different dynamic optimization algorithms to be tested in a standard way in a multi-modal environment. MPB allows the creation of different dynamic fitness functions consisting of a number of peaks with varying location, height and width. The primary measure for performance evaluation is offline error, which is the average difference between the optimum and the best evaluation since the last environment change. Obviously, this value is always a positive number and it is zero only for perfect tracking. Several PSO methods have been developed and tested using MPB such as (Blackwell & Branke, 2004a; Blackwell & Branke, 2004b; Li, Branke, & Blackwell, 2006; Mendes & Mohais, 2005). Particularly Blackwell & Branke (2004a) proposed a successful multi-swarm approach. The idea behind this is that different swarms can converge to different peaks and track them when the environment changes. The swarms interact only by mutual repulsion that keeps any two swarms from converging to the same peak.

Similar to the aforementioned EAs, PSO might exhibit some major problems and severe drawbacks such as parameter dependency (Lovberg & Krink, 2002) and loss of diversity (Riget & Vesterstrom, 2002). Particularly the latter phenomenon increases the probability of being trapped in a local optimum and it is the main source of premature convergence especially when the dimensionality of the search space is large (Van den Bergh, 2002) and the problem to be optimized is multi-modal (Esquivel & Coello, 2003; Riget & Vesterstrom, 2002). Another reason for premature convergence is that particles are flown through a single point, which is (randomly) determined by *gbest* and *pbest* positions and this point is not even guaranteed to be a local optimum (Van den Bergh & Engelbrecht, 2002). Since PSO was proposed for static problems in general, effects of such drawbacks eventually become much more severe for dynamic environments. Various modifications and PSO variants have been proposed in order to address these problems such as (Abraham, Das, & Roy, 2007; Chen & Li, 2007; Chen, Peng, & Jian, 2007; Christopher & Seppi, 2004; Clerc, 1999; Eberhart, Simpson, & Dobbins, 1996; Higashi & Iba, 2003; Ince, Kiranyaz, & Gabbouj, 2009; Janson & Middendorf, 2005; Kaewkamnerdpong & Bentley, 2005; Krohling & Coelho, 2006; Liang & Qin, 2006; Li et al., 2006; Lovberg, 2002; Lovberg & Krink, 2002; Mendes, Kennedy, & Neves, 2004; Peng, Reynolds, & Brewster, 2003; Peram, Veeramachaneni, & Mohan, 2003; Ratnaweera, Halgamuge, & Watson, 2003; Ratnaweera, Halgamuge, & Watson, 2002; Riget & Vesterstrom, 2002; Richards & Ventura, 2003; Shi & Eberhart, 1998; Shi & Eberhart, 2001; Van den Bergh & Engelbrecht, 2002; Van den Bergh & Engelbrecht, 2004; Xie, Zhang, & Yang, 2002a; Xie, Zhang, & Yang, 2002b; Xie, Zhang, & Yang, 2002c; Yasuda, Ide, & Iwasaki, 2003; Zhang & Xie, 2003). Such methods usually try to improve the diversity among the particles and the search mechanism either by changing the update equations towards more diversified versions or adding more randomization to the system (to particle velocities, positions, etc.) or simply resetting some or all of them randomly when some conditions are met. However, their performance improvement might be quite limited even in static environments and most of them use *more* parameters and/or thresholds to accomplish this whilst making the PSO variant even more parameter dependent. Therefore, they do not yield set a reliable solution for dynamic environments, which usually have a multi-modal nature and high dimensionality.

Another major drawback of the basic PSO and the aforementioned variants is that they can only be applied to a search (solution) space with a fixed dimensionality. However, in many optimization problems, the optimum dimension is also unknown (e.g. data clustering, object extraction, optimization of the dynamic functions, etc.) and should thus be determined within the PSO process. Take for example the color-based *image* segmentation as a data clustering application, where the optimum dimension of the solution space corresponds to the true number of clusters (segments) in the data (color) space, which cannot be known beforehand. In such a case the PSO process should perform a multi-dimensional search in order to determine both the true (optimum) number of clusters and the optimum centroid location for each cluster. The problem becomes even more challenging when it is applied over a dynamic environment such as a *video* where both the number of clusters (segments) and their centroids (dominant colors) are changing over time. Yet since the change between consecutive frames is not drastic but rather minor, instead of performing a new clustering in the color domain via multi-dimensional search for each frame in the video, for a new (next) frame, the PSO process can establish a follow-up mechanism to track the optimum (number of) clusters (segments) from the previous frame. Therefore, using the past history about global and local optima becomes a crucial information to search for the current optima (both dimension and location).

In this paper, we shall first introduce a recent technique that significantly improves the global convergence performance of PSO by forming an artificial Global Best particle (aGB) fractionally. This algorithm, the so-called Fractional Global Best Formation (FGBF), collects the best dimensional components from each swarm particle and fractionally creates the *aGB*, which will replace *gbest* as a guide for the swarm, if it turns out to be better than the swarm's native *gbest* particle. We then propose a novel multi-swarm algorithm, which combines multi-swarms with the FGBF technique so that each swarm can apply FGBF distinctively. Via applying the proposed techniques on conventional MPB we shall show that they can find and track the global peak in an efficient way and usually in earlier stages. For the multi-dimensional dynamic environments where the optimum dimension also changes over time, we shall then introduce a multi-dimensional (MD) PSO technique, which re-forms the native structure of swarm particles in such a way that they can make inter-dimensional passes with a dedicated dimensional PSO process. Therefore, in a multi-dimensional search space where the optimum dimension is unknown, swarm particles can seek for both positional and dimensional

optima. This eventually pushes the frontier of optimization problems in dynamic environments towards a *global* search in a multi-dimensional space, where the problems in each dimension are possibly multi-modal and dependent on each other in a certain manner. Since the conventional MPB is created for a unique (fixed) dimensionality, we shall propose multi-dimensional extension of the benchmark in which there exists a unique optimum dimension where the global (highest) peak is located. Also the optimum dimension can change over time within a dimension range. In order to provide a certain degree of dependency among individual dimensions, peaks in different dimensions share the common coordinates of the peak locations. This is basically accomplished by subtracting a penalty term, whose magnitude depends on a dimensional error, from the landscape height in non-optimal dimensions. As a result, over the extended MPB, MD PSO can seek for both the optimum dimension and the global peak on it simultaneously. FGBF is an add-on to both multi-swarms and MD PSO which can enhance their performance. We shall show when the multi-dimensional search is needed, the best performance is achieved by their mutual operation. In recent works, both MD PSO and FGBF have been successfully applied over *static* problems such as general data clustering (Kiranyaz, Ince, Yildirim, & Gabbouj, 2010) and evolutionary artificial neural networks (Ince et al., 2009), respectively.

The rest of the paper is organized as follows. Section 2 surveys related work on PSO and MPB. The proposed techniques, MD PSO, multi-swarms with FGBF and their applications over the (extended) MPB are presented in detail in Section 3. Section 4 provides the experiments conducted and discusses the results. Finally, Section 5 concludes the paper.

2. Related work

2.1. The basic PSO algorithm

In the basic PSO method, (*bPSO*), a swarm of particles flies through an N-dimensional search space where each particle represents a potential solution to the optimization problem. Each particle with index a in the swarm, $\xi = \{x_1, \ldots, x_a, \ldots, x_S\}$, is represented by the following characteristics:

$x_{a,j}(t)$: jth dimensional component of the position of particle a, at time t.
$v_{a,j}(t)$: jth dimensional component of the velocity of particle a, at time t.
$y_{a,j}(t)$: jth dimensional component of the personal best (*pbest*) position of particle a, at time t.
$\hat{y}_j(t)$: jth dimensional component of the global best position of the swarm, at time t.

Let f denote the fitness function to be optimized. Without loss of generality assume that the objective is to find the maximum of f in an N-dimensional space. Then the personal best of particle a can be updated at iteration t as,

$$y_{a,j}(t) = \begin{cases} y_{a,j}(t-1) & \text{if } f(x_a(t)) < f(y_a(t-1)) \\ x_{a,j}(t) & \text{else} \end{cases} \quad j = 1, 2, \ldots, N \tag{1}$$

Since *gbest* is the index of the GB particle, $\hat{y}(t) = y_{gbest}(t)$. Then at each iteration in a PSO process, positional updates are performed for each dimensional component, $j \in [1,N]$ and for each particle, $a \in [1,S]$, as follows:

$$\begin{aligned} v_{a,j}(t+1) &= w(t)v_{a,j}(t) + c_1 r_{1,j}(t)(y_{a,j}(t) - x_{a,j}(t)) \\ &\quad + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{a,j}(t)) \\ x_{a,j}(t+1) &= x_{a,j}(t) + v_{a,j}(t+1) \end{aligned} \tag{2}$$

where w is the inertia weight, (Shi & Eberhart, 1998) and c_1, c_2 are the acceleration constants which are usually set to 1.49 or 2. $r_{1,j} \sim U(0,1)$ and $r_{2,j} \sim U(0,1)$ are random variables with uniform distribution. Recall from the earlier discussion that the first term in the summation is the *memory* term, which represents the contribution of the previous velocity over the current velocity, the second term is the *cognitive* component, which represents the particle's own experience and the third term is the *social* component through which the particle is "guided" by the *gbest* particle towards the GB solution so far obtained. Although the use of inertia weight, w, was later added by Shi & Eberhart (1998), into the velocity update equation, it is widely accepted as the basic form of PSO algorithm. A larger value of w favors exploration while a small inertia weight favors exploitation. As originally introduced, w is often linearly decreased from a high value (e.g. 0.9) to a low value (e.g. 0.4) during iterations of a PSO run. Depending on the problem to be optimized, PSO iterations can be repeated until a specified number of iterations, say *IterNo*, is exceeded, velocity updates become zero, or the desired fitness score is achieved (i.e. $f > \varepsilon_C$). Accordingly the general pseudo-code of the *bPSO* can be given as follows:

bPSO (*termination criteria*: {*IterNo*, ε_C, ...}, $V_{\max}$)

1. For $\forall a \in [1,S]$ do:
   1.1 Randomize $x_a(1)$, $v_a(1)$
   1.2 Let $y_a(0) = x_a(1)$
   1.3 Let $\hat{y}(0) = x_a(1)$
2. End For.
3. For $\forall t \in [1, IterNo]$ do:
   3.1 For $\forall a \in [1,S]$ do:
      3.1.1 Compute $y_a(t)$ using (1)
      3.1.2 If $(f(y_a(t)) > \max(f(\hat{y}(t-1), f(y_i(t)))_{1 \le i < a})$ then *gbest* = a and $\hat{y}(t) = y_a(t)$
   3.2 End For.
   3.3 If any *termination criterion* is met, then **Return**.
   3.3 For $\forall a \in [1,S]$ do:
      3.4.1 For $\forall j \in [1,N]$ do:
         3.4.1.1 Compute $v_{a,j}(t+1)$ using (2)
         3.4.1.2 If $(|v_{a,j}(t+1)| > V_{\max})$ then clamp it to $|v_{a,j}(t+1)| = V_{\max}$
         3.4.1.3 Compute $x_{a,j}(t+1)$ using (2)
      3.4.2 End For.
   3.5 End For.
4. End For.
5. **Return.**

Velocity clamping also called "dampening" with the user-defined maximum range $V_{\max}$ (and $-V_{\max}$ for the minimum) as in Step 3.4.1.2 is one of the earliest attempts to control or prevent oscillations. Such oscillations are indeed crucial since they broaden the search capability of the swarm; however, they have a potential drawback of oscillating continuously around the optimum point. Therefore, such oscillations should be dampened and the convergence is achieved with the proper use of the velocity clamping and the inertia weight. Furthermore, this is the *bPSO* algorithm where the particle *gbest* is determined within the entire swarm. Another major topological approach, the so-called *lbest*, also exists where the swarm is divided into overlapping neighborhoods of particles and instead of defining *gbest* and $\hat{y}(t) = y_{gbest}(t)$ over the entire swarm, for a particular neighborhood N_i, the (local) best particle is referred as *lbest* with the position $\hat{y}_i(t) = y_{lbest}(t)$. Neighbors can be defined with respect to particle indices (i.e. $i \in [j-l, \ldots j+l]$ or by using some other topological forms (Suganthan, 1999). It is

obvious that *gbest* is a special case of *lbest* scheme where the neighborhood is defined as the entire swarm. The *lbest* approach is one of the earlier attempts, which usually improves the diversity; however, it is slower than the *gbest* approach. PSO variants for dynamic environments will be reviewed in Section 2.3.

2.2. Moving Peaks Benchmark

Conceptually speaking, MPB developed by Branke (2008), is a simulation of a configurable dynamic environment changing over time. The environment consists of a certain number of peaks with varying locations, heights and widths. The dimensionality of the fitness function is fixed in advance and thus is an input parameter of the benchmark. An N-dimensional fitness function with m peaks is expressed as,

$$F(\vec{x},t) = \max\left(B(\vec{x}), \max_{p=1\ldots m} P(\vec{x}, h_p(t), w_p(t), \vec{c}_p(t))\right) \quad (3)$$

where $B(\vec{x})$ is a time variant basis landscape, whose utilization is optional, and P is the function defining the height of the pth peak at location $\vec{x}$, where each of the m peaks can have its own dynamic parameters such as height, $h_p(t)$, width, $w_p(t)$ and location vector of the peak center, $\vec{c}_p(t)$. Each peak parameter can be initialized randomly or set to a certain value and after a time period (number of evaluations), T_e, at time (evaluation) t, a change over a single peak, p, can be defined as follows:

$$\begin{aligned} h_p(t) &= h_p(t-T_e) + r\Delta h \\ w_p(t) &= w_p(t-T_e) + r\Delta w \\ \vec{c}_p(t) &= \vec{c}_p(t-T_e) + \vec{v}_p(t) \end{aligned} \quad (4)$$

where $r \sim U(0,1)$, Δh and Δw are the heights and width change severities, respectively, and $\vec{v}_p(t)$ is a shift vector, which is a linear combination of a random vector and the previous shift vector, $\vec{v}_p(t-T_e)$. The shift vector $\vec{v}_p(t)$ is always normalized to length *vlength*, which is called change severity. Accordingly, the shift vector $\vec{v}_p(t)$ can be defined as

$$\vec{v}_p(t) = vlength\frac{(1-\lambda)\vec{r}(t) + \lambda\vec{v}_p(t-T_e)}{\|(1-\lambda)\vec{r}(t) + \lambda\vec{v}_p(t-T_e)\|},$$

where λ is the correlation factor, which defines the level of location change randomness. The types and number of peaks along with their initial heights and widths, environment (search space) dimension and size, change severity, level of change randomness and change frequency can be defined. To facilitate standard comparative evaluations among different algorithms, three standard settings of such MPB parameters, the so-called *Scenarios*, have been defined. *Scenario* 2 is the most widely used and it allows a range of values, among them the following are commonly used: number of peaks = 10, change severity *vlength* = 1.0, correlation value $\lambda = 0.0$ and peak change frequency = 5000. In *Scenario* 2 no basis landscape is used and the peak type is a simple *cone* with the following expression,

$$P(\vec{x}, h_p(t), w_p(t), \vec{c}_p(t)) = h_p(t) - s_p(t)\|\vec{x} - \vec{c}_p(t)\| \text{ where}$$
$$s_p(t) = \frac{h_p(t)}{w_p(t)} \text{ and } \|\vec{x} - \vec{c}_p(t)\| = \sqrt{\sum_{i=1}^{N}(x_i - c_{pi})^2} \quad \forall x_i \in \vec{x},\ \forall c_{pi} \in \vec{c}_p(t) \quad (5)$$

where $s_p(t)$ is the slope and $\|.\|$ is the *Euclidean* distance between two N-dimensional vectors, $\vec{x}$ and $\vec{c}_p(t)$. More detailed information on MPB and the rest of the parameters used in this benchmark can be obtained from Branke (2008).

2.3. Multi-swarm PSO

The main problem of using the basic PSO algorithm in a dynamic environment is that eventually the swarm will converge to a single peak – whether global or local. When another peak becomes the global maximum as a result of an environmental change, it is likely that the particles keep circulating close to the peak to which the swarm has converged and thus they cannot find the new global maximum. Blackwell and Branke have addressed this problem in Blackwell & Branke (2004a, 2004b) by introducing multi-swarms that are actually separate PSO processes. Each particle is now a member of one of the swarms only and it is unaware of other swarms. The main idea is that each swarm can converge to a separate peak. Swarms interact only by mutual repulsion that keeps them from converging to the same peak. For a single swarm it is essential to maintain enough diversity so that the swarm can track small location changes of the peak to which it is converging. For this purpose Blackwell and Branke introduced charged and quantum swarms, which are analogues to an atom having a nucleus and charged particles randomly orbiting it. The particles in the nucleus take care of the fine tuning of the result while the charged particles are responsible of detecting the position changes. However, it is clear that, instead of charged or quantum swarms, some other method can also be used to ensure sufficient diversity among particles of a single swarm so that the peak can be tracked despite of small location changes.

As one might expect, the best results are achieved when the number of swarms is set equal to the number of peaks. However, it is then required that the number of peaks is known beforehand. Blackwell (2007) presents self-adapting multi-swarms, which can be created or removed during the PSO process and, therefore, it is not necessary to fix the number of swarmsbeforehand.

The repulsion between swarms is realized by simply re-initializing the worse of the two swarms if they move within a certain range from each other. Using physical repulsion could lead to equilibrium, where swarm repulsion prevents both swarms from getting close to a peak. A proper limit closer to which the swarms are not allowed to move, r_{rep} is attained by using the average radius of the peak basin, r_{bas}. If p peaks are evenly distributed in X^N, $r_{rep} = r_{bas} = X/p^{1/N}$. Detailed information about multi-swarms can be obtained in Blackwell & Branke (2004a, 2004b).

3. The proposed optimization technique for dynamic environments

In this section, we first introduce the multi-swarms with FGBF technique and its application to MPB. The multi-dimensional extension of PSO, the so-called MD PSO, will be detailed next. Finally we show their mutual application over the (extended) MPB.

3.1. FGBF technique

For those problems where dimensional fitness evaluation is possible, Fractional Global Best Formation (FGBF) can be used to avoid the pre-mature convergence by providing a significant diversity obtained from a proper *fusion* of the swarm's best components (the individual dimension(s) of the current position of each particle in the swarm). Some problems such as data clustering, on the other hand, can exhibit some kind of non-trivial interdependency among the dimensional components of the search space. For these cases, a proper *approximation* for the fractional fitness evaluation, if possible, should be designed to perform FGBF. In either case, FGBF fractionally creates an artificial GB particle, called aGB, at each iteration in a PSO process by selecting the best particle (dimensional) components from the entire swarm. Therefore, especially during the initial steps, aGB can be and, most of the time, is a better alternative than the native *gbest* particle since it has the advantage of assessing each dimension of every particle in the swarm individually, and forming the *aGB* particle fractionally by using the best

components among them. This process naturally uses the available diversity among individual dimensional components and thus it can prevent the swarm from being trapped in local optima due to its ongoing and ever-varying particle creations. At each iteration FGBF is performed after the assignment of the swarm's *gbest* particle (i.e. performed between Steps 3.2 and 3.3 in the pseudo-code of *bPSO*) and, if *aGB* turns out to be better than *gbest*, the personal best location of the *gbest* particle is replaced by the location of the *aGB* particle and, since $\hat{y}(t) = y_{gbest}(t)$, the artificially created particle is thus used to guide the swarm through the *social* component in (2). In other words, the swarm will be guided only by the best (winner) between the native *gbest* and *aGB* particle at any time. In the next iteration, a new *aGB* particle is created and it will again compete against the personal best of *gbest* (which can be also a former *aGB* now).

Suppose that for a swarm ξ, FGBF is performed in a PSO process in dimension *N*. Recall from the earlier discussion that in a particular iteration, *t*, each PSO particle, *a*, has the following components: position ($x_{a,j}(t)$), velocity ($v_{a,j}(t)$) and the personal best position ($y_{a,j}(t)$), $j \in [1,N]$. As the *aGB* particle is fractionally (re-) created from the individual dimensions of some swarm particles at each iteration, it does not need the velocity term and, therefore, it does not have to remember its personal best location.

Let $f(a,j)$ be the dimensional fitness score of the *j*th component of the position of particle *a* and $f(gbest,j)$ be the dimensional fitness score of the *j*th component of the personal best position of the *gbest* particle. Suppose that all dimensional fitness scores ($f(a,j)$, $\forall a \in [1,S]$ and $f(gbest,j)$) can be computed in Step 3.1 and FGBF can then be plugged in between Steps 3.2 and 3.3 of *bPSO*'s pseudo-code. Accordingly, the pseudo-code for FGBF can be expressed as follows:

FGBF in *bPSO* (ξ,$f(a,j)$)

1. Let $a[j] = \arg\max_{a \in \xi, j \in [1,N]} (f(a,j))$ be the index of particle yielding the maximum $f(a,j)$ for the *j*th dimensional component.
2. $x_{aGB,j}(t) = x_{a[j],j}(t)$ for $\forall j \in [1,N]$
3. If $f(gbest,j) > f(a[j],j)$ then $x_{aGB,j}(t) = y_{gbest,j}(t)$
4. If $(f(x_{aGB}(t)) > f(y_{gbest}(t)))$ then $y_{gbest}(t) = x_{aGB}(t)$ and $\hat{y}(t) = x_{aGB}(t)$
5. **Return**.

Note that Step 1 along with the computation of $f(a,j)$ depends entirely on the optimization problem. It keeps track of partial fitness contributions from each individual dimensional component of each particle's position (the potential solution). Take for instance the function minimization problem as illustrated in Fig. 1 where 2D space is used for illustration purposes. In the figure, three particles in a swarm are ranked as the 1st (or the *gbest*), the 3rd and the 8th with respect to their proximity to the target position (or the global solution) of some function. Although *gbest* particle (i.e. 1st rank particle) is the closest in the overall sense, the particles ranked 3rd and 8th provide the best *x* and *y* dimensions (closest to the target's respective dimensions) in the entire swarm and hence the *aGB* solution via FGBF yields a better (closer) particle than the swarm's native *gbest*. Particularly in Kiranyaz et al. (2010), the usage and merits of FGBF for the optimization of several multi-modal, non-linear (benchmark) functions in high dimensions have been shown where all MD PSO runs with FGBF found the global minimum at the target dimension for all runs, over all functions, regardless of the dimension, swarm size and modality, and without any exception. This is the case where the function

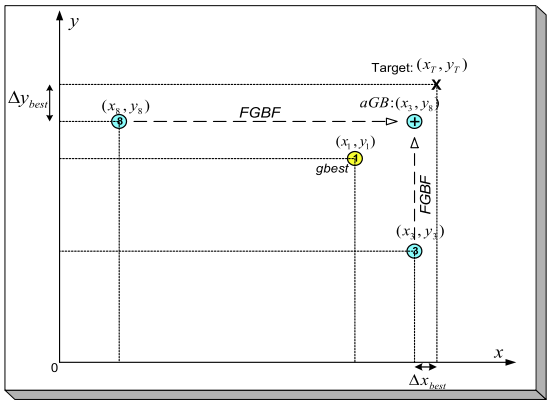


Fig. 1. A sample FGBF operation in 2D space.

to be optimized is known *a priori* and FGBF is easier to use. For those problems where the knowledge of how each fractional dimension contributes to the overall fitness is no longer available (i.e. black-box problems), FGBF can still be adapted with respect to the problem, e.g. in Kiranyaz et al. (2010), see the usage of FGBF over general data clustering where the optimum dimension of the search space (number of clusters) is also unknown and hence MD PSO with the proper application of FGBF is used to find the optimum (number of) clusters in a data space.

3.2. FGBF application for MPB

The previous section introduced the principles of FGBF theory within a *bPSO* process in a single dimension and referred to some of its applications in other domains, each of which is in a static environment. However, in dynamic environments this approach eventually leads the swarm to converge to a single peak (whether global or local) and therefore, it may lose its ability to track other peaks. As any of the peaks can become the optimum peak as a result of environmental changes, it is likely to lead to a suboptimal solution. This is the basic reason of utilizing the multi-swarms along with the FGBF operation. As described in Section 2.3, the mutual repulsion between swarms is performed and for computing the distance between two swarms, we use the distance between the swarms' global best locations. Instead of charged or quantum swarms, FGBF is the mechanism alternatively used to provide necessary diversity and thus to enable peak tracking if peaks' location are changed. We also re-initialize the particle velocities after each environment change to further contribute to the diversity.

A particle with index *a* in a swarm ξ, represents a potential solution and therefore, the *j*th component of an *N*-dimensional point (x_j, $j \in [1,N]$) is stored in its positional component, $x_{a,j}(t)$, at a time *t*. The aim of the PSO process is to search for the global optimum point, which maximizes $P(\vec{x}, h_p(t), w_p(t), \vec{c}_p(t))$, in other words, finding the global (highest) peak in MPB environment. Recall that in *Scenario* 2 of MPB the peaks used are *cone* shaped, as given in (5). Since in (5), $h_p(t)$ and $s_p(t)$ are both set by MPB, finding the highest peak is equivalent to minimizing the $\|\vec{x} - \vec{c}_p(t)\|$ term, yielding $f(a,j) = -(x_j - c_{pj})^2$. Step 3.1 in *bPSO*'s pseudo-code computes the (dimensional) fitness scores ($f(a,j)$, $f(gbest,j)$) of the *j*th components ($x_{a,j}$, $y_{gbest,j}$) and in Step 2 of the FGBF process, the dimensional component yielding the maximum $f(a,j)$ is then placed in *aGB*. In Step 3 these dimensional components are replaced by dimensional components of the personal best position of the *gbest* particle, if they yield higher dimensional fitness scores. We do not expect that

dimensional fitness scores can be evaluated with respect to the optimum peak since this requires the *a priori* knowledge of the global optimum, instead we use either the *current* peak where the particle resides on or the peak to which the swarm is converging (swarm peak). We shall thus consider and evaluate both *modes* separately.

3.3. MD PSO algorithm

Instead of operating in a fixed dimension N, the MD PSO algorithm is designed to seek both positional and dimensional optima within a dimension range, ($D_{\min} \leqslant d \leqslant D_{\max}$). In MD PSO each particle has two sets of components, each of which is subjected to one of two independent and consecutive processes. The first one is the regular positional PSO, i.e. the traditional velocity updates and due positional shifts in a d-dimensional search space. The second one is the dimensional PSO, which allows the particle to navigate through dimensions. Accordingly, each particle keeps track of its last position, velocity and personal best position (*pbest*) in a particular dimension so that when it re-visits the same dimension at a later time, it can perform its regular "positional" fly using this information. The dimensional PSO process of each particle may then move the particle to another dimension where it will remember its earlier positional status and keep "flying" within the positional PSO process in this dimension, and so on. The swarm, on the other hand, keeps track of the *gbest* particles in all dimensions, each of which respectively indicates the best (global) position so far achieved and can thus be used in the regular velocity update equation for that dimension. Similarly the dimensional PSO process of each particle uses its personal best dimension in which the personal best fitness score has so far been achieved. Finally, the swarm keeps track of the global best dimension, *dbest*, among all the personal best dimensions. The *gbest* particle in *dbest* dimension represents the optimum solution and dimension, respectively.

The MD PSO process at time (iteration), is represented by the following characteristics:

- $xx_{a,j}^{xd_a(t)}(t)$: jth component (dimension) of the position of particle a, in dimension $xd_a(t)$.
- $vx_{a,j}^{xd_a(t)}(t)$: jth component (dimension) of the velocity of particle a, in dimension $xd_a(t)$.
- $xy_{a,j}^{xd_a(t)}(t)$: jth component (dimension) of the personal best (*pbest*) position of particle a, in dimension $xd_a(t)$.
- *gbest (d)*: Global Best particle index in dimension d.
- $x\hat{y}_j^d(t)$: jth component (dimension) of the global best position of swarm, in dimension d.
- $xd_a(t)$: current dimension of particle a.
- $v\tilde{d}_a(t)$: dimensional velocity of particle a.
- $x\tilde{d}_a(t)$: personal best dimension of particle a.

Fig. 2 shows sample MD PSO and *bPSO* particles with indices a. *bPSO* particle that is at a (fixed) dimension, $N = 5$, contains only positional components whereas MD PSO particle contains both positional and dimensional components, respectively. In the figure the dimension range for the MD PSO is between 2 and 9, therefore the particle contains eight sets of positional components. In this example the current dimension where the particle with index a resides is 2 ($xd_a(t) = 2$) whereas its personal best dimension is 3 ($x\tilde{d}_a(t) = 3$). Therefore, at time t, a positional PSO update is first performed over the positional elements, $xx_a^2(t)$ and then the particle may move to another dimension with respect to the dimensional PSO. Recall that each positional element, $xx_{a,j}^2(t), j \in \{1,2\}$, represents a potential solution in the data space of the problem.

Let f denote the fitness function that is to be optimized within a certain dimension range, $[D_{\min}, D_{\max}]$. In accordance with the scope of the current work and without loss of generality assume that the objective is to find the maximum (position) of f in the optimum dimension within a multi-dimensional search space. Assume that the particle a visits (back) the same dimension after T iterations (i.e. $xd_a(t) = xd_a(t+T)$), then the personal best position can be updated in iteration $t+T$ as follows,

$$xy_{a,j}^{xd_a(t+T)}(t+T) = \begin{cases} xy_{a,j}^{xd_a(t)}(t) \\ \text{if } f\left(xx_a^{xd_a(t+T)}(t+T)\right) < f\left(xy_a^{xd_a(t)}(t)\right) \\ xx_{a,j}^{xd_a(t+T)}(t+T) \\ \text{else} \end{cases}$$
$$j = 1, 2, \ldots, xd_a(t+T) \tag{6}$$

To overcome the necessity to remember the iteration numbers, when a particle has visited a certain dimension last time, the following updates are performed for each dimensional component except $xd_a(t)$, i.e. $j \in [1,d]$ where $d \in [D_{\min}, D_{\max}] - \{xd_a(t)\}$ and for each particle, $a \in [1,S]$,

$$xy_{a,j}^d(t) = xy_{a,j}^d(t-1), x\hat{y}_j^d(t) = x\hat{y}_j^d(t-1), \quad \text{for } \forall d \in [D_{\min}, D_{\max}] - \{xd_a(t)\} \tag{7}$$

Furthermore, the personal best dimension of particle a can be updated in iteration $t+1$ as follows,

$$x\tilde{d}_a(t) = \begin{cases} x\tilde{d}_a(t-1) & \text{if } f\left(xx_a^{xd_a(t)}(t)\right) < f\left(xy_a^{x\tilde{d}_a(t-1)}(t-1)\right) \\ xd_a(t) & \text{else} \end{cases} \tag{8}$$

Recall that *gbest (d)* is the index of the Global Best particle in dimension d and so $x\hat{y}^d(t) = xy_{gbest(d)}^d(t)$. For a particular iteration t, and for a particle with index $a \in [1,S]$, first the positional components are updated in its current dimension, $xd_a(t)$ and then the dimensional

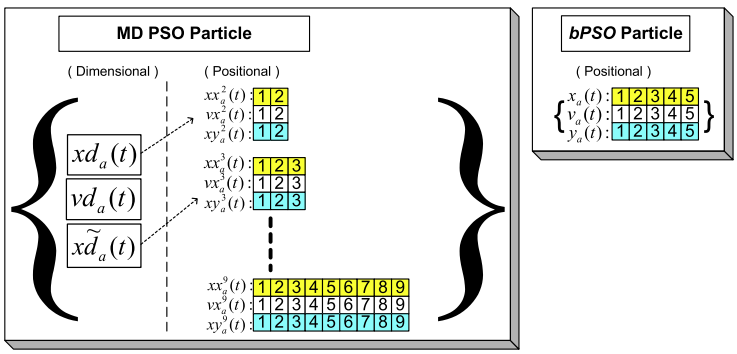


Fig. 2. Sample MD PSO (right) vs. *bPSO* (left) particle structures. For MD PSO $\{D_{\min} = 2, D_{\max} = 9\}$ and at the current time t, $xd_a(t) = 2$ and $x\tilde{d}_a(t) = 3$. For *bPSO* $N = 5$.

update is performed to determine its next dimension, $xd_a(t+1)$. The positional update is performed for each dimension component, $j \in [1, xd_a(t)]$, as follows:

$$\begin{aligned} vx_{a,j}^{xd_a(t)}(t+1) &= w(t)\, vx_{a,j}^{xd_a(t)}(t) + c_1 r_{1,j}(t)\left(xy_{a,j}^{xd_a(t)}(t) - xx_{a,j}^{xd_a(t)}(t)\right) \\ &\quad + c_2 r_{2,j}(t)\left(x\hat{y}_{j}^{xd_a(t)}(t) - xx_{a,j}^{xd_a(t)}(t)\right) \\ xx_{a,j}^{xd_a(t)}(t+1) &= xx_{a,j}^{xd_a(t)}(t) + C_{vx}\left[vx_{a,j}^{xd_a(t)}(t+1), \{V_{\min}, V_{\max}\}\right] \\ xx_{a,j}^{xd_a(t)}(t+1) &\leftarrow C_{xx}\left[xx_{a,j}^{xd_a(t)}(t+1), \{X_{\min}, X_{\max}\}\right] \end{aligned} \tag{9}$$

where $C_{xx}[..] \equiv C_{vx}[..]$ are the clamping operators applied over each positional component, $xx_{a,j}^{d}$ and $vx_{a,j}^{d}$. $C_{vx}[..]$ may or may not be applied depending on the optimization problem but $C_{vx}[..]$ is basically needed to avoid exploding. Each operator can be applied in two different ways, such as,

$$C_{xx}\left[xx_{a,j}^{d}(t), \{X_{\min}, X_{\max}\}\right] = \begin{Bmatrix} xx_{a,j}^{d}(t) & \text{if } X_{\min} \leqslant xx_{a,j}^{d}(t) \leqslant X_{\max} \\ X_{\min} & \text{if } xx_{a,j}^{d}(t) < X_{\min} \\ X_{\max} & \text{if } xx_{a,j}^{d}(t) > X_{\max} \end{Bmatrix} \tag{10.a}$$

$$C_{xx}\left[xx_{a,j}^{d}(t), \{X_{\min}, X_{\max}\}\right] = \begin{Bmatrix} xx_{a,j}^{d}(t) & \text{if } X_{\min} \leqslant xx_{a,j}^{d}(t) \leqslant X_{\max} \\ U(X_{\min}, X_{\max}) & \text{else} \end{Bmatrix} \tag{10.b}$$

where option (a) is a simple thresholding to the range limits and (b) re-initializes randomly the jth positional component ($j \leqslant d$).

Note that the particle's new position, $xx_{a}^{xd_a(t)}(t+1)$, will still be in the same dimension, $xd_a(t)$; however, the particle *may* jump to another dimension afterwards. Therefore, for all the other dimensions in the dimensional range except $xd_a(t)$, i.e. $\forall d \in \{D_{\min}, \ldots, D_{\max}\} - \{xd_a(t)\}$, the following updates are necessary for each dimensional component, $j \in [1, \ldots, d]$ and for each particle, $a \in [1, S]$:

$$\begin{aligned} &vx_{a,j}^{d}(t+1) = vx_{a,j}^{d}(t), xx_{a,j}^{d}(t+1) = xx_{a,j}^{d}(t) \\ &\text{for} \forall d \in [D_{\min}, D_{\max}] - \{xd_a(t)\} \end{aligned} \tag{11}$$

The dimensional updates are computed with the following equations:

$$\begin{aligned} vd_a(t+1) &= \lfloor vd_a(t) + c_1 r_1(t)(\tilde{xd}_a(t) - xd_a(t)) \\ &\quad + c_2 r_2(t)(dbest - xd_a(t)) \rfloor \\ xd_a(t+1) &= xd_a(t) + C_{vd}[vd_a(t+1), \{VD_{\min}, VD_{\max}\}] \\ xd_a(t+1) &\leftarrow C_{xd}[xd_a(t+1), \{D_{\min}, D_{\max}\}] \end{aligned} \tag{12}$$

where $\lfloor . \rfloor$ is the *floor* operator, C_{xd} [..] and $C_{vd}[..]$ are the clamping operators applied over dimensional components, $xd_a(t)$ and $vd_a(t)$, respectively. As in (2), we employ the inertia weight for positional velocity update; however, we have witnessed no benefit of using it for dimensional PSO, and hence we left it out of (12) for the sake of simplicity. $C_{vd}[..]$ is similar to $C_{vx}[..]$, which is basically applied to avoid exploding and we use the basic thresholding for this, expressed as follows:

$$C_{vd}[vd_a(t), \{VD_{\min}, VD_{\max}\}] = \begin{Bmatrix} vd_a(t) & \text{if } VD_{\min} \leqslant vd_a(t) \leqslant VD_{\max} \\ VD_{\min} & \text{if } vd_a(t) < VD_{\min} \\ VD_{\max} & \text{if } vd_a(t) > VD_{\max} \end{Bmatrix} \tag{13}$$

$C_{xd}[..]$, on the other hand, is a mandatory clamping operator, which keeps the dimensional jumps within the dimension range of the problem, [$D_{\min}$, $D_{\max}$]. Furthermore within $C_{xd}[..]$, an *optional* inflow buffering mechanism can also be implemented. This can be a desired property, which allows only sufficient number of particles in a certain dimension and thus avoids the redundancy. Particularly *dbest* and dimensions within the close proximity have a natural attraction and without such buffering mechanism, the majority of swarm particles may be hosted within this local neighborhood and hence other dimensions might encounter a severe depletion. To prevent this, the buffering mechanism should control the in-flow of the particles (by the dimensional velocity updates) to a particular dimension. On some early *bPSO* implementations over problems with low (and fixed) dimensions, 15–20 particles were usually sufficient for a successful operation. However, in high dimensions this may not be so since more particles are usually needed as the dimension increases. Therefore, we empirically set the limit to be proportional to the solution space dimension and not less than 15. At time t, let $P_d(t)$ be the number of particles in dimension d. $C_{xd}[..]$ can then be expressed with the (optional) buffering mechanism as follows:

$$C_{xd}[xd_a(t), \{D_{\min}, D_{\max}\}] = \begin{Bmatrix} xd_a(t-1) & \text{if } P_{xd_a(t)}(t) \geqslant \max(15, xd_a(t)) \\ xd_a(t-1) & \text{if } xd_a(t) < D_{\min} \\ xd_a(t-1) & \text{if } xd_a(t) > D_{\max} \\ xd_a(t) & \text{else} \end{Bmatrix} \tag{14}$$

In short, the clamping and buffering operator, $C_{xd}[..]$, allows a dimensional jump only if the target dimension is within dimensional range and has space for a newcomer. Accordingly, the general pseudo-code of the MD PSO method can be expressed as follows:

MD PSO (*termination criteria*: {*IterNo*, ε_C, ...})

1. For $\forall a \in [1, S]$ do:
   1.1. Randomize $xd_a(1)$, $vd_a(1)$
   1.2. Initialize $\tilde{xd}_a(0) = xd_a(1)$
   1.3. For $\forall d \in \{D_{\min}, D_{\max}\}$ do:
      1.3.1. Randomize $xx_a^d(1)$, $xv_a^d(1)$
      1.3.2. Initialize $xy_a^d(0) = xx_a^d(1)$
      1.3.3. Initialize $x\hat{y}^d(0) = xx_a^d(1)$
   1.4. End For.
2. End For.
3. For $\forall t \in [1, IterNo]$ do:
   3.1. For $\forall a \in [1, S]$ do:
   3.1.1. If $\left(f\left(xx_a^{xd_a(t)}(t)\right) > f\left(xy_a^{xd_a(t)}(t-1)\right)\right)$ then do:
      3.1.1.1. $xy_a^{xd_a(t)}(t) = xx_a^{xd_a(t)}(t)$
      3.1.1.2. Update $\tilde{xd}_a(t)$ according to (8)
   3.1.2. Else $xy_a^{xd_a(t)}(t) = xy_a^{xd_a(t)}(t-1)$
   3.1.3. End If
   3.1.4. If $\left(f\left(xx_a^{xd_a(t)}(t)\right) > \max\left(f(x\hat{y}^{xd_a(t)}(t-1)), \max_{1 \leqslant p < a}\left(f\left(xx_p^{xd_a(t)}(t)\right)\right)\right)\right)$ then do:
      3.1.4.1. $gbest\ (xd_a(t)) = a$
      3.1.4.2. If $\left(f\left(xx_a^{xd_a(t)}(t)\right) > f(x\hat{y}^{dbest}(t-1))\right)$ then $dbest = xd_a(t)$
   3.1.5. End If
   3.1.6. Do updates in other dimensions according to (7)
   3.2. End For.
   3.3. If the *termination criteria* are met, then **Stop**.
   3.4. For $\forall a \in [1, S]$ do:
   3.4.1. For $\forall j \in [1, xd_a(t)]$ do:
      3.4.1.1. Compute $vx_{a,j}^{xd_a(t)}(t+1)$ and $xx_{a,j}^{xd_a(t)}(t+1)$ using (9)
      3.4.1.2. Update velocities and locations in other dimensions using (11)
   3.4.2. End For.
   3.4.3. Compute $vd_a(t+1)$ and $xd_a(t+1)$ using (12)
   3.5 End For.
4. End For.

Once the MD PSO process terminates, the optimum solution will be $x\hat{y}^{dbest}$ in the optimum dimension, *dbest*, achieved by the particle with the index *gbest (dbest)* and finally the best (fitness) score achieved will naturally be $f(x\hat{y}^{dbest})$. Note that MD PSO is *only* a natural extension or a generic form of the basic PSO, as it searches for both optimum dimension and solution (in the optimum dimension). Note that this is not a 'superiority' in terms of *convergence*, rather the ability of searching the optimum solution space dimension while searching for the positional optimum, in a simultaneous way. In other words, the basic PSO is a MD PSO process in a *fixed* dimension, which basically means that *bPSO* is identical to MD PSO for a particular (fixed) dimension. However, contrary to *bPSO*, due to its ability to simultaneously search for both optimum dimension and solution, MD PSO can conveniently be used in many applications where the optimum dimension is unknown. For instance in Kiranyaz et al. (2010) MD PSO (with FGBF) has been used to find the true number of clusters in a complex data space. This is obviously a tremendous advantage since with a traditional approach such as *K-means*, the number of clusters, *K*, has to be given *a priori* and this may not be possible for many complex problems. In another work (Ince et al., 2009), MD PSO has been used for the automatic design of Artificial Neural Networks (ANNs) by evolving to the optimal network configuration(s) for a particular problem. Thus MD PSO can seek for the positional optimum in the error space and dimensional optimum in the architecture space. The optimum dimension converged at the end of a MD PSO process corresponds to the best ANN configuration (or the most appropriate ANN for that problem) where the trained network parameters (connections, weights and biases) can then be resolved from the positional optimum reached in that dimension. This presents a significant advantage over many traditional training methods such as the well-known back-propagation algorithm, which can only be used to train a single ANN with a fixed configuration.

3.4. The proposed techniques over multi-dimensional MPB

For testing the proposed multi-dimensional optimization technique, we extended Branke's MPB into a multi-dimensional version, in which there exists many search space dimensions within a dimensional range $[D_{\min}, D_{\max}]$, and the optimal dimension changes over time in addition to the dynamic nature of the conventional MPB. Peak locations in different dimensions share the common peak center coordinates and thus such an extension further allows exploitation of the information gathered in other search space dimensions.

The multi-dimensional extension of the MPB is simple. The initialization and changes of peak locations must now be done in the highest possible search space dimension, $D_{\max}$. Locations in the other dimensions can be obtained simply by leaving out the redundant coordinates (non-existing dimensions). The optimal dimension is chosen randomly every time the environment is changed. Therefore, the fitness function with m peaks in multi-dimensional environment can be expressed as,

$$F(\vec{x}^d, t) = \max\left(B(\vec{x}^d), \max_{p=1\ldots m} P\left(\vec{x}^d, d, h_p(t), w_p(t), \vec{c}_p^d(t)\right)\right) \tag{15}$$

where $d \in [D_{\min}, D_{\max}]$ is the dimension of position $\vec{x}^d$ and $\vec{c}_p^d(t)$ refers to the first d coordinates (dimensions) of the peak center location. A *cone* peak is now expressed as follows:

$$P(\vec{x}^d, h_p(t), w_p(t), \vec{c}_{pd}(t)) = h_p(t) - s_p(t) * \left\|\vec{x}^d - \vec{c}_p^d(t)\right\| / d - (D_{opt} - d)^2 \quad \text{where}$$
$$s_p(t) = \frac{h_p(t)}{w_p(t)} \quad \text{and} \quad \left\|\vec{x}^d - \vec{c}_p^d(t)\right\|$$
$$= \sqrt{\sum_{i=1}^{d}\left(x_i^d - c_{pi}^d\right)^2} \quad \forall x_i^d \in \vec{x}^d, \forall c_{pi}^d \in \vec{c}_p^d(t) \tag{16}$$

where D_{opt} is the current optimal dimension. If compared with expression (5), now for all non-optimal dimensions a penalty term $(D_{opt} - d)^2$ is subtracted from the whole environment. In addition to that the peak slopes are scaled by the term $1/d$. The purpose of this scaling is to prevent the benchmark from favoring the lower dimensions. Otherwise a solution, whose every coordinate differs from the optimum by 1.0 would be a lot better solution in a lower dimension as the *Euclidian* distance is used.

Similar to the uni-dimensional (PSO) case, each positional component $xx_a^d(t)$ of MD PSO particle represents a potential solution in dimension d. The only difference is that now the dimensionality of the optimal solution is not known beforehand, but it can vary within the defined range. Even a single particle can provide potential solutions in different dimensions as it makes inter-dimensional passes as a result of MD PSO process. Our dynamic multi-dimensional optimization algorithm combines multi-swarms and FGBF with MD PSO. As in the different dimensions the common coordinates of the peak locations are the same, it does not seem purposeful for two swarms to converge to the same peak in different dimensions. Therefore, the mutual repulsion between swarms is extended to affect swarms that are in different dimensions. Obviously, only the common coordinates are considered when the swarm distance is computed.

FGBF naturally exploits information gathered in other dimensions. When the *aGB* particle is created, FGBF algorithm is not limited to use dimensional components from only those particles which are in a certain dimension, but it can combine dimensional coordinates of particles in different dimensions. Note that as we still use the dimensional fitness score, $f(a,j) = -(x_j - c_{pj})^2$, the common coordinates of the positional components of the *aGB* particle created in different search space dimensions, $d \in [D_{\min}, D_{\max}]$, shall be the same. In other words, it is not necessary to create the positional components of the *aGB* particle from scratch in every search space dimension $d \in [D_{\min}, D_{\max}]$, instead in dimensions higher than $D_{\min}$, only one (new) coordinate (dimension) to the *aGB* particle is created and added. Note also that it is still possible that in some search space dimensions *aGB* beats the native *gbest* particle, while in other dimensions it does not. In the multi-dimensional version also the dimension and dimensional velocity of each particle are re-initialized after an environmental change in addition to the particle velocities in each dimension.

4. Experimental results

An extensive set of experiments was conducted over both conventional (uni-dimensional) MPB and the extended (multi-dimensional) MPB and the results will be presented in the following subsections.

4.1. Results on conventional MPB

We conducted an exhaustive set of experiments over the MPB *Scenario* 2 using the settings given in Section 2.2. In order to investigate the effect of multi-swarm settings, we used different numbers of swarms and numbers of particles in a swarm. We applied both FGBF modes using the current peaks and the swarm peaks. To investigate how FGBF and multi-swarms individually contribute to the results, we also made experiments without using either of them.

Fig. 3 presents the current error plot, which shows the difference between the global maximum and the current best result during the first 80,000 function evaluations, when 10 swarms each with four particles are used and the swarm peak mode is applied for the FGBF operation. It can be seen from the figure that as the environment changes after every 5000 evaluations, it causes the

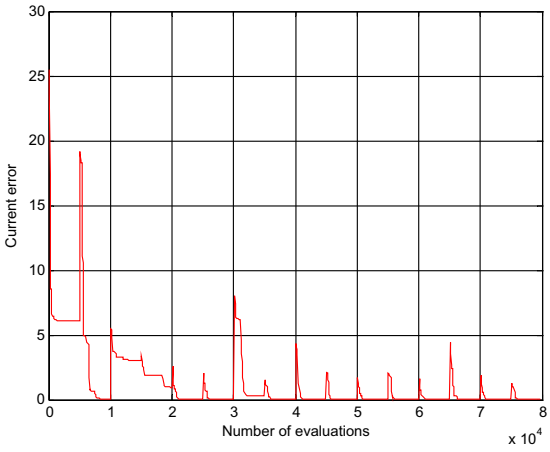


Fig. 3. Current error at the beginning of a run.

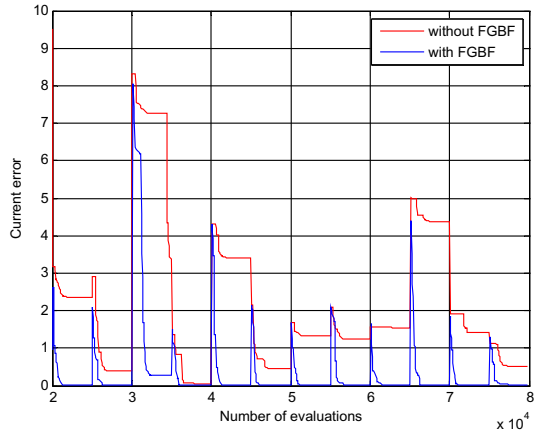


Fig. 5. Effect of FGBF on results.

results to temporarily deteriorate. However, it is clear that after environment changes the results are better than at the very beginning, which shows the benefit of tracking the peaks instead of randomizing the swarm when a change occurs. The figure also reveals other typical features of the algorithms behavior. First of all, after the first few environment changes the algorithm has not yet been behaving as well as later. This is because the swarms have not yet converged to a peak. Generally, it is more difficult to initially converge to a narrow or low peak than to keep tracking a peak that becomes narrow and/or low. It can also be seen that typically the algorithm gets close to the optimal solution before the environment is changed again. In few cases, where the optimal solution is not found, the algorithm has for some reason been unable to keep a swarm tracking that peak, which is too narrow.

In Figs. 4 and 5 the contributions of multi-swarms with FGBF are demonstrated. The algorithm is run on MPB applying the same environment changes, first with both using multi-swarms and FGBF, then without multi-swarms and finally without FGBF. The same settings are used as before. Without multi-swarms the number of particles is set to 40 to keep the total number of particles unchanged.

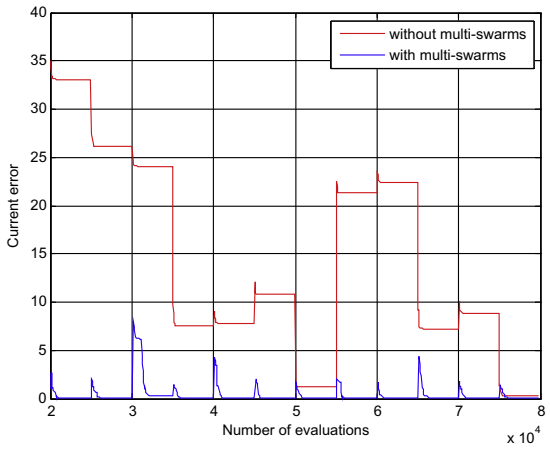


Fig. 4. Effect of multi-swarms on results.

As expected, the results without multi-swarms are significantly deteriorated due to the aforementioned reasoning. When the environment is changed, the highest point of the peak to which the swarm is converging can be found quickly, but that can provide good results only when that peak happens to be the global optimum. When multi-swarms are used, but without using the FGBF, it is clear that the algorithm can still establish some kind of follow-up of peaks as the results immediately after environment changes are only slightly worse than with FGBF. However, if FGBF is not used, the algorithm can seldom find the global optimum. Either there is no swarm converging to the highest peak or the peak center just cannot be found fast enough.

For comparative evaluations, we selected five of the state-of-the-art methods, which use the same benchmark system, the MPB with the same settings. The best MPB results published so far by these competing methods are listed in Table 1.

The overall best results have been achieved by the Extremal Optimization algorithm (Moser & Hendtlass, 2007); however, this algorithm is specially and only designed for MPB and its applicability for other practical dynamic problems is not clear. The best results by a PSO-based algorithm have been achieved by Blackwell and Branke's multi-swarm algorithm described in Section 2.3.

The numerical results of the proposed technique in terms of the offline error are listed in Table 2. Each result given is the average of 50 runs, where each run consists of 500,000 function evaluations. As expected the best results are achieved when 10 swarms are used. Four particles in a swarm turned out to be the best setting. Between the two FGBF modes, better results are obtained when the swarm peak is used instead of the peak closest to each particle.

4.2. Results on multi-dimensional MPB

On the extended MPB we conducted experiments with settings similar to those used in the fixed dimension except that the change

Table 1
Best results on MPB up to date.

Source	Base algorithm	Offline error
Blackwell and Branke (2004a)	PSO	2.16 ± 0.06
Li et al. (2006)	PSO	1.93 ± 0.06
Mendes and Mohais (2005)	Differential evolution	1.75 ± 0.03
Blackwell and Branke (2004b)	PSO	1.75 ± 0.06
Moser and Hendtlass (2007)	Extremal optimization	0.66 ± 0.02

Table 2
Offline error using *Scenario 2*.

No. of swarms	No. of particles	Swarm peak	Current peak
10	2	1.81 ± 0.50	2.58 ± 0.55
10	3	1.22 ± 0.43	1.64 ± 0.53
10	4	**1.03 ± 0.35**	**1.37 ± 0.50**
10	5	1.19 ± 0.32	1.52 ± 0.44
10	6	1.27 ± 0.41	1.59 ± 0.57
10	8	1.31 ± 0.43	1.61 ± 0.45
10	10	1.40 ± 0.39	1.70 ± 0.55
8	4	1.50 ± 0.41	1.78 ± 0.57
9	4	1.31 ± 0.54	1.66 ± 0.54
11	4	1.09 ± 0.35	1.41 ± 0.42
12	4	1.11 ± 0.30	1.46 ± 0.43

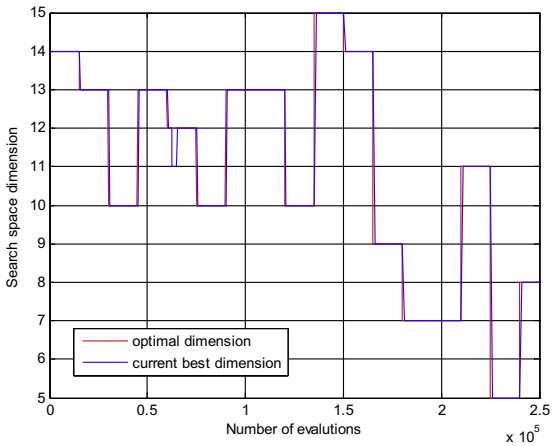


Fig. 6. Optimum dimension tracking in a MD PSO run.

frequency used is set to 15,000. The search space dimension range used is $d \in [5, 15]$. Fig. 6 shows how the global optimal dimension changes over time and how MD PSO is tracking these changes. Current best dimension represents the dimension, where the best solution is achieved among all swarms' *dbest* dimensions. Ten multi-swarms are used with seven particles in each. FGBF is used with the swarm peak mode. It can be seen that the algorithm always finds the optimal dimension, even though the difference in peak heights between the optimal dimension and its neighbor dimensions is quite insignificant (=1) compared to the peak heights (30–70). Fig. 7 shows how the current error behaves during the first 250,000 evaluations, when the same settings are used. It can be seen that the algorithm behavior is similar to the uni-dimensional case, but now the initial converging phase, when the algorithm is not yet behaving at its best is longer. Similarly it takes a longer time to regain the optimal behavior if follow-up of some peaks is lost for some reason (it is, for example, possible that higher peaks hide other lower peaks under them).

Figs. 8 and 9 illustrate the effect of using multi-swarms on the results. Without multi-swarms the number of particles is set to 70. Fig. 8 shows that a single swarm can also find the optimal dimension easily; however, as in the uni-dimensional case, without use of multi-swarms, the optimal peak can be found only if it happens to be the peak to which the swarm is converging. This can be seen in Fig. 9. During the initial converging phase of the multi-swarm algorithm results with and without multi-swarms are similar. This indicates that both algorithms initially converge to the same peak (highest) and as a result of the first few environmental changes some peaks that are not yet discovered by multi-swarms become the highest.

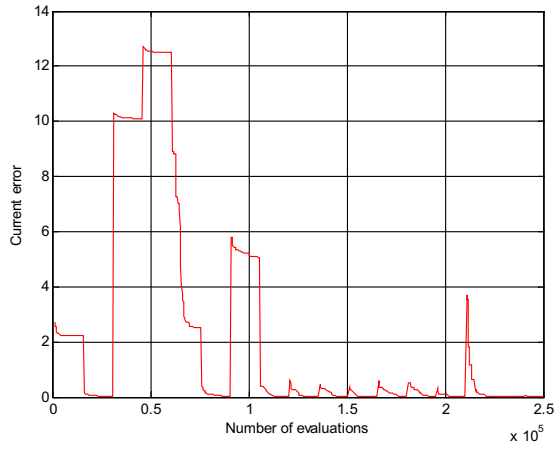


Fig. 7. Current error at the beginning of a MD PSO run.

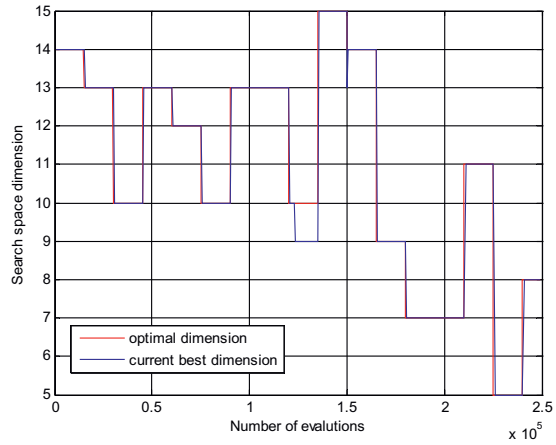


Fig. 8. Optimum dimension tracking without multi-swarms in a MD PSO run.

Figs. 10 and 11 illustrate similarly the effect of FGBF on the results. In Fig. 10 it can be seen that without FGBF the algorithm has severe problems in tracking the optimal dimension. In this case, it loses the benefit of exploiting the natural diversity among the dimensional components and also it is not able to exploit information gathered in other dimensions. Therefore, even if some particles visit the optimal dimension, they cannot track the global peak *fast* enough that they would hence surpass the best results in other dimensions. Therefore, the algorithm gets trapped in some sub-optimum dimension where it happens to find the best results in an early phase. Such reasons also cause the current error to be generally higher without FGBF, as can be seen in Fig. 11.

The numerical results in terms of offline error are given in Table 3. Each result given is the average of 50 runs, where each run consists of 500,000 function evaluations. As in the uni-modal case, best results are achieved when the number of swarms is equal to the number of peaks, which is 10. Interestingly when the swarm peak *mode* is used the optimal number of particles becomes seven

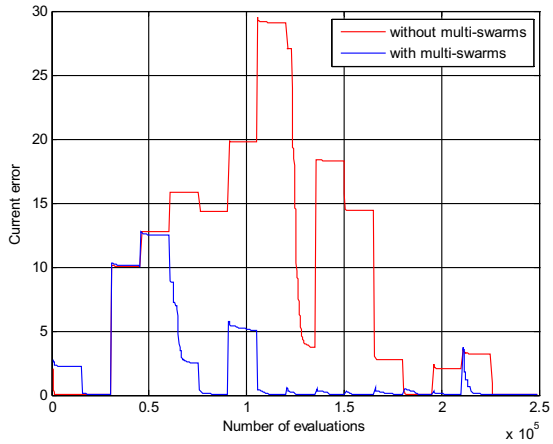


Fig. 9. Effect of multi-swarms on the performance.

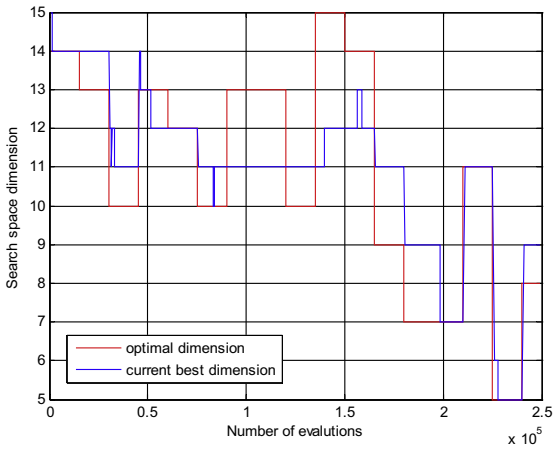


Fig. 10. Optimum dimension tracking without FGBF in a MD PSO run.

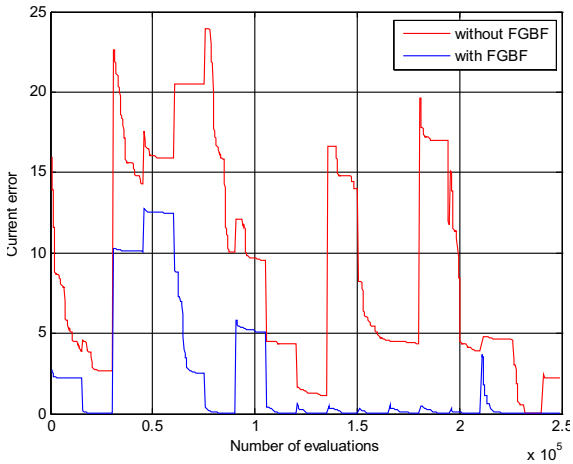


Fig. 11. Effect of FGBF on the performance.

Table 3
Offline error on extended MPB.

No. of swarms	No. of particles	Swarm peak	Current peak
10	4	2.01 ± 0.98	**3.29 ± 1.44**
10	5	1.77 ± 0.83	3.41 ± 1.69
10	6	1.79 ± 0.98	3.64 ± 1.60
10	7	**1.69 ± 0.75**	3.71 ± 1.74
10	8	1.84 ± 0.97	4.21 ± 1.83
10	10	1.96 ± 0.94	4.20 ± 2.03
8	7	1.79 ± 0.91	3.72 ± 1.86
9	7	1.83 ± 0.84	4.30 ± 2.15
11	7	1.75 ± 0.91	3.52 ± 1.40
12	7	2.03 ± 0.97	4.01 ± 1.97

while with the current peak mode it is still four. Note that these results cannot be directly compared with the results on the conventional MPB since the objective function of the multi-dimensional MPB is somewhat different.

5. Conclusions

In this paper, we presented two PSO techniques, namely, FGBF with multi-swarms and MD PSO for an efficient and robust optimization over the dynamic systems. Both MD PSO and FGBF have been successfully used in static optimization problems particularly as a cure to common drawbacks of the family of PSO methods such as *a priori* fixation of the search space dimension and pre-mature convergence to local optima. MD PSO efficiently addresses the former drawback by defining a new particle structure and embedding the ability of dimensional navigation into the core of the process. It basically allows particles to make inter-dimensional 'passes' with a dedicated PSO process whilst performing regular positional updates in every dimension they visit. Although the ability of determining the optimum dimension where the global solution exists is gained with MD PSO, its convergence performance is still limited to the same level as *bPSO*, which suffers from the lack of diversity among particles. This leads to a pre-mature convergence to local optima especially when multi-modal problems are optimized in high dimensions. Realizing that the main problem lies in fact at the inability of using the available diversity among the dimensional components of swarm particles, the FGBF technique adapted in this paper addresses this problem by collecting the best components and fractionally creating an *aGB* particle that has the potential to be a better "guide" than the swarm's native *gbest* particle. On MPB we do not expect to receive fractional scores with respect to the global (highest) peak, but instead we use either the peak, on which the particle is currently located (current peak) or the peak to which the swarm is converging (swarm peak). Especially swarm peak *mode* makes it possible to find and track the global highest peak quite successfully in a dynamic environment.

In order to make comparative evaluations with the current state-of-the-art, FGBF with multi-swarms is then applied over a benchmark system, the MPB. The results over the conventional MPB with common settings used (i.e. *Scenario* 2) clearly indicate the superiority of the proposed technique over other PSO-based methods. To make the benchmark more generic for real-world applications where the optimum dimension can be unknown too, MPB is extended to a multi-dimensional system in which there is a certain amount of dependency among dimensions. Note that without such dependency embedded, the benchmark would be just a bunch of independent MPBs in different dimensions and thus a distinct and independent optimization process would be sufficient for each dimension. Recall that the convergence behavior of both *bPSO* and MD PSO is the same since MD PSO is only an extension of PSO for the multi-dimensional search. The performance of

both methods degrades with the increasing modality and dimensionality due to the reasons mentioned earlier. When performed with FGBF and multi-swarms, MD PSO exhibits both global convergence ability and an impressive speed gain so that their mutual performance surpasses *bPSO* by several magnitudes. The experiments conducted over the extended MPB approve that the proposed MD PSO technique with multi-swarms and FGBF always finds and tracks the optimum dimension where the global peak resides. On both (conventional and extended) MPBs, the proposed techniques generally find and track the global peak, yet they can occasionally converge to a near-optimum peak, particularly if the height difference happens to be insignificant.

Overall, the proposed techniques fundamentally upgrade the particle structure and the swarm guidance, both of which accomplish substantial improvements in terms of speed and accuracy. Both techniques are modular and independent from each other, i.e. one can be performed without the other whilst other PSO methods/variants can also be performed conveniently with (either of) them.

References

Abraham, A., Das, S., & Roy, S. (2007). Swarm intelligence algorithms for data clustering. In *Soft computing for knowledge discovery and data mining book, Part IV* (pp. 279–313).

Angeline, P. J. (1997). Tracking extrema in dynamic environments. In *Proceedings of the 6th conference on evolutionary programming* (pp. 335–345). Springer-Verlag.

Bäck, T. (1998). On the behaviour of evolutionary algorithms in dynamic environments. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 446–451).

Bäck, T., & Kursawe, F. (1995). Evolutionary algorithms for fuzzy logic: A brief overview. *Fuzzy logic and soft computing* (pp. 3–10). Singapore: World Scientific.

Bäck, T., & Schwefel, H. P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolution on Computers, 1*, 1–23.

Blackwell, T. M. (2007). Particle swarm optimization in dynamic environments. *Evolutionary Computation in Dynamic and Uncertain Environments, Studies in Computational Intelligence, 51*, 29–49.

Blackwell, T. M., & Branke, J. (2004a). Multi-swarm optimization in dynamic environments. *Applications of evolutionary computation* (Vol. 3005, pp. 489–500). Springer.

Blackwell, T. M., & Branke, J. (2004b). Multi-swarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation, 10/4*, 51–58.

Branke, J. (2008). Moving Peaks Benchmark. http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/, viewed 26/06/08.

Chen, X., & Li, Y. (2007). A modified PSO structure resulting in high exploration ability with convergence guaranteed. *IEEE Transactions on Systems, Man, and Cybernetics, Part B, 37*(5), 1271–1289.

Chen, Y.-P., Peng, W.-C., & Jian, M.-C. (2007). Particle swarm optimization with recombination and dynamic linkage discovery. *IEEE Transactions on Systems, Man, and Cybernetics, Part B, 37*(6), 1460–1470.

Christopher, K. M., & Seppi, K. D. (2004). The Kalman swarm. A new approach to particle motion in swarm optimization. In *Proceedings of the genetic and evolutionary computation conference* (pp. 140–150). GECCO.

Clerc, M. (1999). The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 3, pp. 1951–1957).

Eberhart, R., & Shi, Y. (2001). Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of computational evolution conference (CEC 2001)* (pp. 94–100). NJ, US.

Eberhart, R., Simpson, P., & Dobbins, R. (1996). *Computational intelligence. PC tools*. Boston, MA, USA: Academic Press, Inc..

Engelbrecht, A. P. (2005). *Fundamentals of computational swarm intelligence*. John Wiley & Sons.

Esquivel, S. C., & Coello, C. A. (2003). On the use of particle swarm optimization with multimodal functions. *IEEE Transactions on Evolutionary Computation, 2*, 1130–1136.

Fayyad, U. M., Shapire, G. P., Smyth, P., & Uthurusamy, R. (1996). *Advances in knowledge discovery and data mining*. Cambridge, MA: MIT Press.

Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.

Higashi, H., & Iba, H. (2003). Particle swarm optimization with gaussian mutation. In *Proceedings of the IEEE swarm intelligence symposium* (pp. 72–79).

Ince, T., Kiranyaz, S., & Gabbouj, M. (2009). A generic and robust system for automated patient-specific classification of electrocardiogram signals. *IEEE Transactions on Biomedical Engineering, 56*(5), 1415–1426.

Janson, S., & Middendorf, M. (2005). A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernetics, Part B, 35*(6), 1272–1282.

Kaewkamnerdpong, B., & Bentley, P. J. (2005). Perceptive particle swarm optimization: An investigation. In *Proceedings of the IEEE swarm intelligence symposium* (pp. 169–176). California.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks* (Vol. 4, pp. 1942–1948). Perth, Australia.

Kiranyaz, S., Ince, T., Yildirim, A., & Gabbouj, M. (2010). Fractional particle swarm optimization in multi-dimensional search space. *IEEE Transactions on Systems, Man, and Cybernetics, Part B, 40*(2), 298–319.

Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, Massachussetts: MIT Press.

Krohling, R. A., & Coelho, L. S. (2006). Coevolutionary particle swarm optimization using gaussian distribution for solving constrained optimization problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B, 36*(6), 1407–1416.

Liang, J. J., & Qin, A. K. (2006). Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions On Evolutionary Computation, 10*(3), 281–295.

Li, X., Branke, J., & Blackwell, T. (2006). Particle swarm with speciation and adaptation in a dynamic environment. In *Proceedings of genetic and evolutionary computation conference* (pp. 51–58). Seattle Washington.

Lovberg, M. (2002). *Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality, MSc thesis*. Department of Computer Science, University of Aarhus, Denmark.

Lovberg, M., & Krink, T. (2002). Extending particle swarm optimisers with self-organized criticality. *Proceedings of the IEEE Congress on Evolutionary Computation, 2*, 1588–1593.

Mendes, R., Kennedy, J., & Neves, J. (2004). The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation, 8*(3), 204–210.

Mendes, R., & Mohais, A. (2005). DynDE: A differential evolution for dynamic optimization problems. *IEEE congress on evolutionary computation*, 2808–2815.

Moser, I., & Hendtlass, T. (2007). A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. *IEEE Congress on Evolutionary Computation*, 252–259.

Omran, M. G., Salman, A., & Engelbrecht, A. P. (2006). *Particle swarm optimization for pattern recognition and image processing*. Berlin: Springer.

Peng, B., Reynolds, R. G., & Brewster, J. (2003). Cultural swarms. *Proceedings of the IEEE Congress on Evolutionary Computation, 3*, 1965–1971.

Peram, T., Veeramachaneni, K., & Mohan, C. K. (2003). Fitness-distance-ratio based particle swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium* (pp. 174–181). IEEE Press.

Ratnaweera, A. C., Halgamuge, S. K., & Watson, H. C. (2002). Particle swarm optimiser with time varying acceleration coefficients. In *Proceedings of the international conference on soft computing and intelligent systems* (pp. 240–255).

Ratnaweera, A. C., Halgamuge, S. K., & Watson, H. C. (2003). Particle swarm optimization with self-adaptive acceleration coefficients. In *Proceedings of the first international conference on fuzzy systems and knowledge discovery* (pp. 264–268).

Richards, M., & Ventura, D. (2003). Dynamic sociometry in particle swarm optimization. In *Proceedings of the sixth international conference on computational intelligence and natural computing* (pp. 1557–1560). North Carolina.

Riget, J., & Vesterstrom, J. S. (2002). *A diversity-guided particle swarm optimizer – the ARPSO, Technical report*. Department of Computer Science, University of Aarhus.

Shi, Y., & Eberhart, R. C. (1998). A modified particle swarm optimizer. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 69–73).

Shi, Y., & Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. *Proceedings of the IEEE congress on evolutionary computation* (Vol. 1, pp. 101–106). IEEE Press.

Suganthan, P. N. (1999). Particle swarm optimiser with neighborhood operator. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 1958–1962). IEEE Press.

Van den Bergh, F. (2002). *An analysis of particle swarm optimizers, PhD thesis*. Department of Computer Science, University of Pretoria, Pretoria, South Africa.

Van den Bergh, F., & Engelbrecht, A. P. (2002). A new locally convergent particle swarm optimizer. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 96–101.

Van den Bergh, F., & Engelbrecht, A. P. (2004). A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation, 3*, 225–239.

Wilson, E. O. (1975). *Sociobiology: The new synthesis*. Cambridge, MA: Belknap Press.

Xie, X., Zhang, W., Yang, Z. (2002). Adaptive particle swarm optimization on individual level. In *Proceedings of the sixth international conference on signal processing* (Vol. 2, pp. 1215–1218).

Xie, X., Zhang, W., & Yang, Z. (2002a). A dissipative particle swarm optimization. *Proceedings of the IEEE Congress on Evolutionary Computation, 2*, 1456–1461.

Xie, X., Zhang, W., & Yang, Z. (2002c). Hybrid particle swarm optimizer with mass extinction. *Proceedings of the International Conference on Communication, Circuits and Systems, 2*, 1170–1173.

Yasuda, K., Ide, A., & Iwasaki, N. (2003). Adaptive particle swarm optimization. *Proceedings of the IEEE international conference on Systems, Man, and Cybernetics, 2*, 1554–1559.

Zhang, W.-J., & Xie, X.-F. (2003). DEPSO: Hybrid particle swarm with differential evolution operator. *Proceedings of the IEEE International Conference on System, Man, and Cybernetics, 4*, 3816–3821.

Publication V

Jenni Raitoharju, Serkan Kiranyaz, Moncef Gabbouj, "Evolutionary Feature Synthesis by Multi-dimensional Particle Swarm Optimization," *5th European Workshop on Visual Information Processing*, pp. 1–6, Dec. 2014.

EVOLUTIONARY FEATURE SYNTHESIS BY MULTI-DIMENSIONAL PARTICLE SWARM OPTIMIZATION

Jenni Raitoharju, Serkan Kiranyaz, Moncef Gabbouj

Tampere University of Technology, Department of Signal Processing,
PL 553, 33101 Tampere, Finland
E-mails: first.last@tut.fi

ABSTRACT

Several existing content-based image retrieval and classification systems rely on low-level features which are automatically extracted from images. However, often these features lack the discrimination power needed for accurate description of the image content and hence they may lead to a poor retrieval or classification performance. This article applies an evolutionary feature synthesis method based on multi-dimensional particle swarm optimization on low-level image features to enhance their discrimination ability. The proposed method can be applied on any database and low-level features as long as some ground-truth information is available. Content-based image retrieval experiments show that a significant performance improvement can be achieved.

Index Terms— Content-based image retrieval, Evolutionary feature synthesis, Multi-dimensional particle swarm optimization

1. INTRODUCTION

In content-based image retrieval (CBIR) systems, features used to describe the image content play a key role. Currently the attention in this research field has been mostly turned toward deep learning, which has indeed provided a fast improvement in quality of available features in the recent years. Furthermore, most efforts on this field are currently headed towards huge online datasets. However, several small-scale image visualization or image search tools are still based on low-level features only. Such systems may be self-implemented and very specialized (e.g. certain types of medical images). If this is the case, pretrained general-purpose deep learning feature extraction approaches will probably fail and retraining is laborious or may not be even possible due to low image volumes. In some cases, the original images are not even available anymore as only the extracted low-level features has been saved and, thus, it not possible to extract better features from the images. In this article, we concentrate on evolutionary feature synthesis (EFS), which is an ideal tool to improve such specialized small-scale systems. It does not need the original images, but operates directly on the extracted low-level features. As long as some labeled training images are available, the proposed EFS can be used as a black-box tool, which takes the original low-level features and transforms them into more discriminative features. From the user point of view, it is not important to understand the details of the EFS. No parameter adjustment is needed and the new obtained features can directly replace the old features in the system.

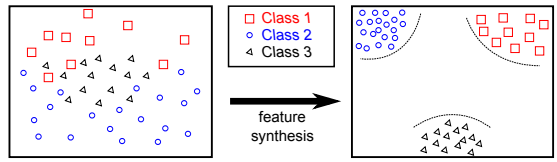


Fig. 1. An illustrative evolutionary feature synthesis, which is applied to 2D feature vectors of a 3-class database

The purpose of the proposed EFS system is to transform the available low-level features so that the discrimination power of the synthesized features is maximized. This will directly improve the results on CBIR or image classification. Fig. 1 illustrates an ideal EFS operation where 2D features of a 3-class database are successfully synthesized in such a way that the discrimination ability of the features has clearly increased.

In this article, we propose an EFS method, which aims to achieve the highest possible discrimination between image features belonging to different classes by simultaneously

1. performing an optimal feature selection,
2. searching for optimal arithmetic, linear, or non-linear, operators,
3. searching for optimal weights for each selected feature,
4. searching for the optimal output feature vector dimensionality,
5. using any given fitness function to measure the quality of the solution.

If the target system requires the feature vectors to have a fixed dimensionality, it is also possible to omit the fourth bullet

and still efficiently exploit all the other properties. The proposed EFS system has been previously successfully applied on audio features [1]. An initial version of the proposed EFS has been also applied on image features in [2], but the fitness functions were not generic nor applicable in real life as they required classification or retrieval over the whole EFS dataset for every fitness evaluation.

The rest of the paper is organized as follows. Related work on feature synthesis is considered in Section 2. The proposed EFS system is introduced in Section 3. Experimental results are given in Section 4. Finally, Section 5 concludes the paper and discusses topics for future work.

2. RELATED WORK ON EVOLUTIONARY FEATURE SYNTHESIS

Most existing methods specifically designed for feature synthesis are based on genetic programming (GP). These methods synthesize new features by applying a series of composite operators on the original low-level features. The composite operators, which are also the individuals of GP are represented by binary trees whose internal nodes are primitive operators and leave nodes are primitive features. The goal of the composite operators is to map the original feature space to a low-dimensional synthesized feature space, in which the items belonging to the same class form a Gaussian component no matter how these images are distributed in the original feature space. The fitness of the composite operators is typically evaluated in terms of the classification accuracy (in the training set) of a Bayesian classifier learned simultaneously with the operators. EFS approaches based on these ideas have been applied face expression recognition [3], object recognition in synthetic aperture radar images [4], and image classification and retrieval [5]. The publications report improved results compared to the original low-level features, even though computational limitations force to keep the dimensionality of the synthesized feature vectors low and to consider only a few possible operators.

In a broader sense, besides the methods specially developed for feature synthesis, also well-known classifiers such as artificial neural networks (ANNs) and support vector machines (SVMs) can be thought as a special kind of feature synthesizers. Commonly ANNs used as classifiers take the original feature vectors as inputs and use 1-of-C output encoding (where C is the number of classes). Their output in an optimal case is a vector corresponding to the image class (e.g. the target vector $[1, 0, ..., 0]$ corresponds to first class and so on). Thus ANNs try to learn a specific feature synthesizer that transforms each feature vector in a certain class to one corner of the C-dimensional cube.

Also several state of-the-art techniques in computer vision and image categorization can be considered as feature synthesis methods. Different methods to find compact codes to represent images, such as PiCoDes [6] or semantic hashing [7], generally start with some low-level features and after a training process a different set of features is obtained. Similarly mid-level feature learning [8] transforms low-level features into semantically more meaningful features via a training process. Generally these methods are not aiming to improve arbitrary low-level features but, on the contrary, their success is dependent on carefully selected low-level features. They aim at preserving the discrimination power of the original features while transforming it into a form more suitable for very large-scale image datasets. They may require manual selection of training categories or the training may require a large number of training samples and a computationally expensive training process. In both cases, the idea is to do training with a separate training data and then apply them on similar general-purpose datasets. Thus, it is hardly feasible to apply any of these methods for feature synthesis over arbitrary low-level features extracted from any (small-scale) dataset especially if the dataset is highly specialized.

3. THE PROPOSED EVOLUTIONARY FEATURE SYNTHESIS METHOD

We propose an EFS method based on multi-dimensional particle swarm optimization (MD PSO) and fractional global best formation (FGBF) [9]. These method are extensions of the well-known particle swarm optimization (PSO) paradigm. As such, they are computationally less demanding than GP, which allows to synthesize features with higher dimensionalities. Also more different operators can be considered without exceeding computational resources. Neither MD PSO nor FGBF have parameters that should be separately adjusted for new datasets. MD PSO also allows optimizing over different dimensionalities. In feature synthesis it means that it is possible to simultaneously optimize the dimensionality of the synthesized features.

3.1. Multi-dimensional particle swarm optimization

MD PSO [10] is an extension of the basic PSO algorithm, where particles can search for solutions with different dimensionalities within a given dimensionality range, $\{d_{\min}, ..., d_{\max}\}$. In order to accomplish this, each particle has two sets of components, each of which has been subjected to one of the two independent and consecutive processes. The first one is the regular positional PSO, which takes place in current dimensionality, d_a, of each particle a and the second one is the dimensional PSO, which allows particles to move between dimensionalities. For each dimensionality d, the swarm keeps track of the best global position so far achieved, $\mathbf{y}_S^d$, and each particle a keeps track of its last position $\mathbf{x}_a^d$, velocity $\mathbf{v}_a^d$, and personal best position $\mathbf{y}_a^d$ in that particular dimensionality so that when it re-visits the same dimensionality at a later time, it can perform the regular positional updates using this information. The dimensional PSO process may then again

move the particle to another dimensionality, where it will remember its positional status and will be updated through the positional PSO process, and so on. Similarly to the positional PSO, the dimensional PSO process uses the personal best dimensionality (in which the personal best fitness score so far has been achieved) of each particle, dy_a, and the global best dimensionality, dy_S, to attract the particles toward a better dimensional solution. Finally, the global best solution in dimensionality dy_S, $\mathbf{y}_S^{dy_S}$, represents the optimal solution and dimensionality, respectively.

Similarly to the basic PSO algorithm, each iteration t of MD PSO starts with the computation of fitness scores for each particle a in its current dimensionality d_a. After this, personal best solutions for all particles in their current dimensionalities ($\mathbf{y}_a^{d_a}, \forall a \in \{1, ..., S\}$), personal best dimensionalities for all particles ($dy_a, \forall a \in \{1, ..., S\}$), the global best solution in each dimensionality ($\mathbf{y}_S^d, \forall d \in \{d_{\min}, ..., d_{\max}\}$) and the global best dimension (dy_S) are updated if necessary. Then each particle's position and velocity will be updated in their current dimensionality using (regular) positional PSO updates:

$$\begin{aligned}\mathbf{v}_a^{d_a}(t+1) =& w(t)\mathbf{v}_a^{d_a}(t) + c_1\mathbf{r_1}(t)\left(\mathbf{y}_a^{d_a}(t) - \mathbf{x}_a^{d_a}(t)\right) + \\ & c_2\mathbf{r_2}(t)\left(\mathbf{y}_S^{d_a}(t) - \mathbf{x}_a^{d_a}(t)\right) \\ \mathbf{x}_a^{d_a}(t+1) =& \mathbf{x}_a^{d_a}(t) + \mathbf{v}_a^{d_a}(t+1).\end{aligned} \tag{1}$$

The particle's new position $\mathbf{x}_a^{d_a}(t+1)$ still has the same dimensionality d_a. However, the dimensional PSO process may now throw the particle into another dimensionality. The search will continue from $\mathbf{x}_a^{d_a}(t+1)$ if the particle later returns to dimensionality $d_a(t)$. The dimensional PSO updates closely resemble the positional ones:

$$\begin{aligned}dv_a(t+1) =& \lfloor dv_a(t) + c_1 r_1(t)\left(dy_a(t) - d_a(t)\right) + \\ & c_2 r_2(t)\left(dy_S(t) - d_a(t)\right)\rfloor \\ d_a(t+1) =& d_a(t) + dv_a(t+1),\end{aligned} \tag{2}$$

where dv_a is the dimensional velocity of particle a and $\lfloor \cdot \rfloor$ denotes the floor operator. Further details and MD PSO pseudo-code can be found in [10].

3.2. Fractional global best formation

Both the basic PSO algorithm and its multi-dimensional extension MD PSO suffer from premature convergence to a local optimum particularly when a high dimensional optimization problem with a multi-modal fitness surface is encountered. The premature convergence is mainly caused by a loss of diversity i.e. the particles gather too close to $\mathbf{y}_S$ in an early phase and lose their ability to explore new potential solutions. FGBF [10] that is a plug-in to the (MD) PSO process can efficiently address the premature convergence problem. The main idea of FGBF is to create at every iteration an additional artificial solution $\mathbf{y}_A$ by combining the best individual elements of particles' solutions. This artificial solution is then compared to $\mathbf{y}_S$ and, if it turns out to have a higher fitness value, $\mathbf{y}_A$ will replace $\mathbf{y}_S$ in Eq. (1). If solution $\mathbf{y}_A$ is not better than $\mathbf{y}_S$, the PSO process will proceed as usually. When FGBF is used in combination with MD PSO a separate artificial solution, $\mathbf{y}_A^d$, is created for every dimensionality d within the dimensionality range $\{d_{\min}, ..., d_{\max}\}$ and in every dimensionality $\mathbf{y}_A^d$ solution competes with $\mathbf{y}_S^d$ solution. However, depending on the optimization task, during the formation of $\mathbf{y}_A^d$ it may be possible to combine elements from particle positions with different dimensionalities. The fitness of elements is evaluated using a specific fractional fitness function, whose selection is problem-dependent similar to the selection of traditional fitness score. For further details and FGBF pseudo-code the reader is referred to [10].

3.3. Overview of the proposed feature synthesis system

For each new feature, the proposed evolutionary search technique performs the following steps:

1. selects $K+1$ original (or already synthesized) features, $f_0, ..., f_K$
2. scales the selected features using proper weights, $w_0, ..., w_K$
3. selects K operators, $\Theta_1, ..., \Theta_K$, to be performed over the (selected and scaled) features
4. bounds the results using a non-linear operator (i.e. tangent hyperbolic, *tanh*).

If the application of a specific operator, Θ_i, on features f_a and f_b is denoted as $\Theta_i(f_a, f_b)$ the synthesis formula used to form each new feature may be given as follows:

$$y_j = \tanh(\Theta_K(...\Theta_2(\Theta_1(w_0 f_0, w_1 f_1), w_2 f_2), ...), w_K f_K). \tag{3}$$

In other words, the operator Θ_1, is first applied to the scaled features f_0 and f_1, then operator Θ_2 is applied to the result of the first operation and the scaled feature f_2 and so on, until the last operator Θ_K is applied to the result of the previous operations and the scaled feature f_K. Furthermore, the dimensionality of the synthesized feature vector (i.e the number of features) is optimized simultaneously with the rest of the synthesizer parameters, as a result of MD PSO's dimensional search process.

The proposed EFS can be performed in one or several runs, where each run can further synthesize the features generated from the previous run. The number of runs, R, can be specified in advance or adaptively determined, i.e. runs are carried out until the point where the fitness improvement is no longer significant. The EFS dataset can be the entire image database or a certain sub-set of it where the ground truth is available.

Note that the proposed EFS can be seen as a generalization of ANNs. A single-layer perceptron (SLP) neuron only performs steps 2 and 4 listed above, while no feature selection (step 1) or operator selection (step 3) is applied. A SLP

also sums an additional bias with scaled features. To allow a similar mechanism, we simply initially complement each input feature vector by a constant value of one. When a bias is beneficial, the proposed EFS can select this constant value, scale it, and combine with other scaled features. The output dimensionality of a SLP is fixed unlike in the proposed EFS. Performing several EFS runs leads to a system similar to a multi-layer perceptron (MLP).

3.4. Encoding of the MD PSO particles in EFS

As evident from the explanation of the MD PSO algorithm in Section 3, the particles are encoded in such a way that the position of particle a at a time t, $\mathbf{x}_a^{d_a}(t)$, represents a potential solution in the search space dimensionality, d_a. In the proposed EFS, the search space dimensionality corresponds to the dimensionality, or simply the number of features, of the synthesized feature vector. Each position in that dimensionality, t, $\mathbf{x}_a^{d_a}(t)$, then encapsulates the complete set of parameters (feature selection, weights, and operators) of a feature synthesizer which transforms the input feature vector into a new feature vector in that dimensionality. Accordingly, each element of the position, $\mathbf{x}_{a,j}^{d_a}(t)$, corresponds to a way of synthesizing the j^{th} output feature in the synthesized feature vector. Note that this element should encapsulate the $K+1$ features, $K+1$ weights, and K operators in an encoded form so as to be used to synthesize the corresponding output feature when decoded. Therefore, we encode $\mathbf{x}_{a,j}^{d_a}(t)$ as a $2K+1$ dimensional vector form, with $K+1$ A-type and K B-type elements, which set the corresponding feature synthesizer parameters as follows,

$$\begin{aligned} f_i &= \lfloor A_i \rfloor + 1, i \in 0, ..., K \\ w_i &= A_i - \lfloor A_i \rfloor, i \in 0, ..., K \\ \Theta_j &= \lceil B_j \rceil, j \in 1, ..., K, \end{aligned} \tag{4}$$

The ranges for the A- and B-type elements can now be set according to number of features in the input feature vector, F, and the number of operators available, T, i.e. $A_i \in [0, F[$ and $B_j \in]0, T]$. The weights, w_i, are thus limited to the range, $0 \leq w_i < 1$.

A sample encoding scheme of MD PSO particles is illustrated in Fig. 2, where the position of particle a in dimensionality 6, $\mathbf{x}_a^6(t)$, is shown. The corresponding feature synthesizer will produce 6-dimensional synthesized feature vector$FV(i)$ and each element of the particle position, $\mathbf{x}_{a,j}^6(t)$, dictates the synthesis of the corresponding j^{th} feature in the output feature vector. The figure shows in detail the synthesis of the first feature dictated by $\mathbf{x}_{a,1}^6(t)$, while the synthesis of the other elements is similarly encoded in the corresponding elements, $\mathbf{x}_{a,2}^6(t)$,...,$\mathbf{x}_{a,6}^6(t)$. In the figure, K is set to 3. If the i^{th} feature of the input feature vector is denoted as $f_{[i]}$, the formula for the synthesis of the first output feature can be now given as

$$y_j = \tanh(\min((w_0 f_{[8]} + w_1 f_{[3]}), w_2 f_{[5]}) * w_3 f_{[3]}). \tag{5}$$

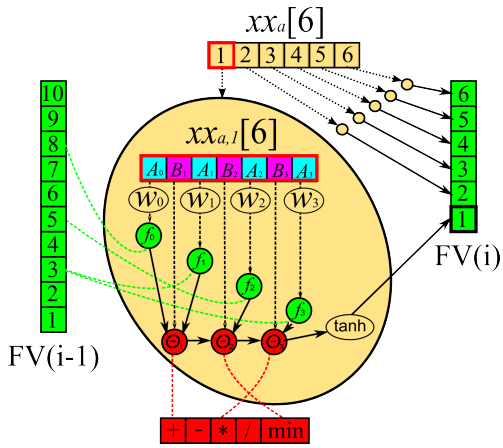


Fig. 2. An illustrative example of particle encoding where its current position is in dimensionality 6 and K is set to 3.

Note that fixing $K+1 = F$, each feature selection as $f_i = f_{[i]}$, and each operator Θ_j to operator "+" makes the proposed feature synthesis technique equivalent to a SLP. Similarly, if more than one EFS runs are performed ($R > 1$), the overall scheme resembles a typical MLP. In short, feed-forward ANNs are indeed a special case of the proposed EFS technique, yet the most complex one due to the use of all input features ($K+1 = F$), which voids the feature selection. Moreover, it is the most limited case, since it uses only a single operator among many possibilities.

3.5. Fitness evaluation

In evaluating the fitness of the feature synthesizers represented by particle positions, we apply a method similar to the one used in ANNs, i.e. we assign target vectors for synthesis of the features from each class, let the EFS system search for a proper synthesis to get to this desired output, and then evaluate the fitness in terms of the mean square error (MSE) between the synthesized output vectors and the target output vectors. However, we do not want to fix the output dimensionality as in ANNs, but instead let the EFS system search for an optimal output dimensionality. Therefore, we generate target output vectors for all dimensionalities within the range of $\{D_{\min}, ..., D_{\max}\}$. For C classes the target vectors first have the 1-of-C section similar to ANN. In generation of the remaining target vector elements, we try to achieve the two criteria for a good error correcting output code (ECOC) suggested in [11], i.e. large row and column separation.

The following procedure is used to generate the target vectors:

1. Assign $b_{\min}$ as the minimum number of bits needed to represent C classes.
2. Form a bit table with $b_{\min}$ rows where each row is the binary representation of the row number.

3. Assign the first $b_{\min}$ target vector values for each class c_i equal to the i^{th} row in the bit table.
4. Move the first row of the bit table to the end of the table and shift the other rows up by one row.
5. Repeat the previous two steps until $D_{\max}$ target vector values have been assigned.
6. Replace the first C values in each target vector by a 1-of-C coded section.
7. Replace each 0 by -1 in the target vectors.

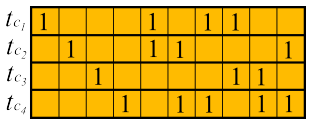


Fig. 3. A sample target vector encoding for 4 classes.

The target output vector generation for a 4-class case is illustrated in Fig. 3. For the sake of clarity the elements set to -1 are shown as empty boxes. $D_{\max}$ is set to 10 and for 4 classes $b_{\min}$ is 2.

The fitness of the particle position $\mathbf{x}_a^{d_a}(t)$ is evaluated as

$$f(\mathbf{x}_{a,j}^{d_a}(t)) = \frac{1}{C}\sum_{j=1}^{C}\sum_{k=1}^{C}\sum_{\forall y\in c_k}(t_{jc_k}-y_j)^2 + \frac{1}{(d-C)^{\alpha}}\sum_{j=C+1}^{d}\sum_{k=1}^{C}\sum_{\forall y\in c_k}(t_{jc_k}-y_j)^2 \quad (6)$$

where t_{jc_k} and y_j denote the j^{th} elements of the target output vector for class c_k and the synthesized output vector. The fractional fitness score of a particle element $\mathbf{x}_{a,j}^{d_a}(t)$, needed for the FGBF process, is simply evaluated as

$$f(\mathbf{x}_{a,j}^{d_a}(t)) = \sum_{k=1}^{C}\sum_{\forall y\in c_k}(t_{jc_k}-y_j)^2 . \quad (7)$$

A detailed explanation on reasons for using such fitness evaluation can be found in [1].

4. EXPERIMENTAL RESULTS

The database used in the experiments contains 1000 medium resolution (384*256 pixels) images obtained from Corel image collection covering 10 classes (natives, beach, architecture, busses, dinosaurs, elephants, roses, horses, mountains, and food). In order to demonstrate the efficacy of the proposed EFS technique for CBIR, we used well-known low-level descriptors that have a limited discrimination power and a severe deficiency for a proper content description. The descriptors used are 64-bins unit normalized RGB and YUV color histograms, 57-D Local Binary Pattern (LBP) and 48-D Gabor texture descriptors. The synthesis depth, K, was set to 7 meaning that only 7 operators and 8 features were used to compose each element of the new feature vector. The parameter α in (6) was set to 1.1. The set of 18 empirically selected operators given in Table 1 was used within the EFS.

Table 1. The set of 18 operators used in the proposed EFS

Θ_j	Formula	Θ_j	Formula
1	$-A$	10	$A*B$
2	$-B$	11	$10(A*B)$
3	$\max(A,B)$	12	A/B
4	$\min(A,B)$	13	$\sin(100\pi(A+B))$
5	$A*A$	14	$\cos(100\pi(A+B))$
6	$B*B$	15	$\tan(100\pi(A*B))$
7	$A+B$	16	$\tan(100\pi(A+B))$
8	$10(A+B)$	17	$0.5*\exp(-(A-B)*(A-B))$
9	$A-B$	18	$0.5*\exp(-(A+B)*(A+B))$

In all the experiments MD PSO parameters were set as follows: The swarm size, S, was set to 200 and the number of iterations, $iterNo$, to 2000. The positional search space range, $\{X_{\min},...,X_{\max}\}$, was set according to the number of operators in use, T, and the number of features in the input feature vectors, F. The positional velocity range, $\{V_{\min},...,V_{\max}\}$, was empirically set to $\{-(X_{\max}-X_{\min})/10,...,(X_{\max}-X_{\min})/10\}$. The dimensionality range,$\{D_{\min},...,D_{\max}\}$, was set to $\{11,...,40\}$ and the dimensional velocity range $\{DV_{\min},...,DV_{\max}\}$, to $\{-4,...,4\}$. All the EFS results are given as the average of 10 separate repetitions of the experiment.

To evaluate the effect of the proposed EFS, we compare the performances obtained using the original and synthesized features on CBIR and image classification. In all cases 45% of the dataset was used to evolve the EFS. CBIR results are evaluated over the whole dataset, while for classification the same 45% is used to train a K-nearest neighbors classifier and the results are evaluated over the remaining 55%. CBIR results for the original features are given in Table 2 and for the synthesized features in Table 3. In Table 3 results are given after a single EFS run and by repeating the EFS process until the results are no longer improved. The average dimension of the synthesized features is also give along with the average number of iterations needed to obtain the best result. For comparison, we also used a 3-layer MLPs for feature synthesis (Table 4). The MLPs are trained using the MD PSO algorithm [12] ($R_{\min}=\{N_i,8,4,N_o\}$, $R_{\max}=\{N_i,16,8,N_o\}$, $iterNo=3*2000=6000$) and the results are obtained using the best architecture found. The classification results over the original features and features synthesized by a single run of the proposed EFS are given in Table 5.

The results clearly indicate a crucial improvement in both CBIR and classification performances. The classification result has been computed over a test set which is separate from the EFS set. The improved result shows that the learned feature synthesis can successfully generalize and be applied

Table 3. CBIR results using features synthesized by the proposed EFS

Corel	RGB(1)	RGB(n)	YUV(1)	YUV(n)	LBP(1)	LBP(n)	Gabor(1)	Gabor(n)	all(1)	all(n)
ANMRR	0.500	0.385	0.505	0.397	0.519	0.396	0.520	0.428	0.357	0.280
AP	0.478	0.596	0.477	0.584	0.465	0.589	0.464	0.559	0.619	0.694
dim.	35.7	12.4	34.7	17.8	35.5	11.2	27.0	11.0		
iter.		19.6		18.7		22.8		24.5		

Table 2. CBIR results using the original low-level features

Corel	RGB	YUV	LBP	Gabor	all
ANMRR	0.589	0.577	0.635	0.561	0.504
AP	0.391	0.405	0.349	0.417	0.473

Table 4. CBIR results using features synthesized by MLPs

Corel	RGB	YUV	LBP	Gabor	all
ANMRR	0.442	0.558	0.545	0.547	0.348
AP	0.543	0.433	0.475	0.442	0.633

on unseen samples. Also it is evident that multiple runs of the proposed EFS can further improve the results. The feature vector dimension decreases during repeated runs, which shows that the proposed EFS can also compress the essential information. The comparison against MLP shows that except for RGB histograms, the retrieval performance statistics for features synthesized by the proposed EFS even with a single run are better than the ones achieved by the best MLP. This basically demonstrates the significance of the feature and operator selection.

5. CONCLUSIONS

In this paper, we proposed an evolutionary feature synthesis technique, which can be used to improve the discrimination power of any the low-level features as long as some ground truth information is available. The proposed EFS simultaneously performs feature selection and an evolutionary search for optimal weights (for the selected features), linear/nonlinear operators (to transform the scaled features) and the optimal output feature vector dimensionality, all in an interleaved way. The quality of the EFS can be ensured by a proper fitness function designed with respect to the main objective of the feature synthesis operation. The proposed technique does not have critical parameters or thresholds that may significantly affect the performance. Experimental results demonstrate that the proposed system can significantly improve the discrimination power of the original features and thus achieve a crucial improvement in classification and CBIR performances. To further improve the performance, different feature synthesizers may be evolved for different classes, since the features essential for discrimination of a certain class may vary. This will be the research topic of our future work.

Table 5. Test classification error using the originall features and features synthesized by a single run of the proposed EFS

Corel	RGB	YUV	LBP	Gabor
Original	0.420	0.371	0.560	0.433
Synthesized	0.312	0.307	0.422	0.367

6. REFERENCES

[1] T. Makinen, S. Kiranyaz, J. Raitoharju, and M. Gabbouj, "An evolutionary feature synthesis approach for content-based audio retrieval," *EURASIP J. on audio, speech and music process.*, , no. 23, Sep. 2012.

[2] S. Kiranyaz, J. Pulkkinen, T. Ince, and M. Gabbouj, "Multi-dimensional evolutionary feature synthesis for content-based image retrieval," in *Proc. 18th IEEE Int. Conf. on Image Processing*, Brussels, Belgium, Sep. 2011.

[3] J. Yu and B. Bhanu, "Evolutionary feature synthesis for facial expression recognition," *Pattern Recognition Lett.*, vol. 27, no. 11, pp. 1289–1298, Aug. 2006.

[4] Y. Lin and B. Bhanu, "Evolutionary feature synthesis for object recognition," *IEEE Trans. Syst., Man, Cybern. Part C, Appl. Rev.*, vol. 35, no. 2, pp. 156–171, 2005.

[5] R. Li, B. Bhanu, and A. Dong, "Feature synthesized em algorithm for image retrieval," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 2, pp. 10:1–10:24, May 2008.

[6] A. Bergamo, L. Torresani, and A. Fitzgibbon, "Picodes: Learning a compact code for novel-category recognition," in *Advances in Neural Information Processing Systems 24*, 2011, pp. 2088–2096.

[7] Y.-G. Jiang, J. Wang, X. Xue, and S.-F. Chang, "Query-adaptive image search with hash codes," *IEEE Trans. Multimedia*, vol. 15, no. 2, pp. 442–453, Feb. 2013.

[8] R. Mittelman, H. Lee, B. Kuipers, and S. Savarese, "Weakly supervised learning of mid-level features with beta-bernoulli process restricted boltzmann machines," in *Proc. IEEE Conf.on Computer Vision and Pattern Recognition*, June 2013, pp. 476–483.

[9] M. Gabbouj S. Kiranyaz, T. Ince, *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*, Springer-Verlag Berlin Heidelberg, 2014.

[10] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Fractional particle swarm optimization in multidimensional search space," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 2, pp. 298–319, 2010.

[11] T.G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *J. Artif. Int. Res.*, vol. 2, no. 1, pp. 263–286, Jan. 1995.

[12] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Evolutionary artificial neural networks by multi-dimensional particle swarm optimization," *Neural Networks*, vol. 22, no. 10, pp. 1448 – 1462, 2009.

Publication VI

Jenni Raitoharju, Serkan Kiranyaz, Moncef Gabbouj, "Feature Synthesis for Image Classification and Retrieval via One-against-all Perceptrons," *Neural Computing and Applications*, pp. 1–15, 2016.

Neural Comput & Applic
DOI 10.1007/s00521-016-2504-4

ORIGINAL ARTICLE

Feature synthesis for image classification and retrieval via one-against-all perceptrons

Jenni Raitoharju[1] · **Serkan Kiranyaz**[2] · **Moncef Gabbouj**[1]

Received: 1 April 2014 / Accepted: 20 July 2016

Abstract Most existing content-based image retrieval and classification systems rely on low-level features which are automatically extracted from images. However, often these features lack the discrimination power needed for accurate description of the image content, and hence, they may lead to a poor retrieval or classification performance. We propose a novel technique to improve low-level features which uses parallel one-against-all perceptrons to synthesize new features with a higher discrimination power which in turn leads to improved classification and retrieval results. The proposed method can be applied on any database and low-level features as long as some ground-truth information is available. The main merits of the proposed technique are its simplicity and faster computation compared to existing feature synthesis methods. Extensive simulation results show a significant improvement in the features' discrimination power.

Keywords Content-based image retrieval and classification · Feature synthesis · Multi-dimensional particle swarm optimization · Multi-layer perceptrons

✉ Jenni Raitoharju
first.last@tut.fi

[1] Department of Signal Processing, Tampere University of Technology, PO Box 527, 33101 Tampere, Finland

[2] Electrical Engineering Department, College of Engineering, Qatar University, Doha, Qatar

1 Introduction

Most existing content-based image retrieval (CBIR) and classification systems rely on low-level features automatically extracted from the images. These features should be discriminative enough to enable highest possible distinction among images belonging to different classes. A lot of work has been done to develop more discriminative features, but it is a well-known fact that there is no such feature extractor that could automatically extract features always matching the human visual perception of the image similarity, since two images belonging to the same class may be visually different and only higher level understanding of the image content can reveal that they should be classified into a common class. This is generally known as the semantic gap problem, and different ways to narrow this gap have been under extensive research in recent years. One of the most obvious ways is to gather knowledge of human perception of image similarity directly from the users. User labeling of the images may be exploited, for example, for feature selection among the vast number of available feature extraction techniques or for feature synthesis to generate a new set of features better matching the human visual perception.

In feature selection, the original features are not changed, but a particular subset of them is selected to be used in content-based image retrieval or classification. So no matter how efficient the feature selection method may be, the final outcome is nothing but a subset of the original features and may still lack the discrimination power needed for an efficient retrieval. In feature synthesis, a linear and/or nonlinear transformation is applied to the original features to synthesize new features. Even if a very limited set of possible transformations is considered, there are such a huge number of possibilities

Published online: 29 July 2016

that it is not feasible to go through all of them. The search space most probably also contains many local optima. Therefore, most existing feature synthesis methods are based on evolutionary algorithms (EAs) [1] such as genetic algorithm (GA) [9] and genetic programming (GP) [18]. The common point of all EAs is that they are stochastic population-based optimization methods that can avoid being trapped in a local optimum. Thus, they can find the optimal solutions; however, this is never guaranteed. The most critical drawback of EA-based methods is their computational complexity, which severely limits the number of problems to which they can be practically applied. Their other common drawback is that they attempt to find a synthesizer which can simultaneously discriminate all the classes from each other. When the number of classes grows, finding a good solution capable of doing that soon becomes an impossible problem for any optimization method.

In this paper, we propose a new approach to feature synthesis where the original features are transformed using parallel one-against-all perceptrons. The proposed method is simple and significantly faster than EA-based methods, and the obtained results still show a significant improvement in both retrieval and classification performances indicating an increase in the features' discrimination ability. The one-against-all topology of the proposed method divides the problem of finding an efficient synthesizer simultaneously discriminating all the classes into several easier problems targeting to discriminate a single class from the rest. At the end of the process, the outputs of each one-against-all perceptron are concatenated into a single feature vector.

The proposed method is generic and applicable to any set of low-level features without requiring manual selection or tuning from the user. This property makes it desirable for those applications where mid-level feature extraction is not a viable option and/or only a certain set of features is available, while the original data is either missing or incomplete. If desired or if there is a reason to assume that a certain classifier is better for a certain type or distribution of data, the proposed approach can be also easily exploited with another one-against-all classifier type besides the perceptrons.

The rest of the paper is organized as follows: Related work on feature synthesis is discussed in Sect. 2. The proposed feature synthesis system is introduced in Sect. 3. Section 4 concentrates on feature synthesis experiments conducted using the proposed method and gives retrieval and classification results along with the comparative evaluations. Finally, Sect. 5 concludes the paper and discusses topics for future work.

2 Related work on feature synthesis

Feature synthesis is still in its infancy as there are only few successful methods proposed up to date. Most existing feature synthesis systems are based on GP [18]. In [3] and [27], GP is utilized to synthesize features for face expression recognition. The individuals are composite operators represented by binary trees whose internal nodes are primitive operators and leaf nodes are primitive features. The primitive features are generated by filtering the original images using a Gabor filter bank at 4 scales and 6 orientations (i.e., 24 images per an original image), and the primitive operators are selected among 37 different options. The fitness of the composite operators is evaluated in terms of the classification accuracy (in the training set) of a Bayesian classifier learned simultaneously with the composite operator. Finally, the best composite operator found is used to synthesize a feature vector for each image in the database and the corresponding Bayesian classifier is then used to classify the image into one of 7 expressions types. The expression recognition rate was only slightly improved compared to similar classification methods where no feature synthesis was applied.

In [22], co-evolutionary genetic programming (CGP) is used to synthesize features for object recognition in synthetic aperture radar (SAR) images. The approach is similar to one in [3, 27], but separate sub-populations are utilized to produce several composite operators. On the other hand, the primitive features used in this application are only 1-dimensional properties computed from the images and thus each composite operator only produces a single 1-dimensional composite feature. The final feature vector is formed by combining the composite features evolved by different sub-populations. Although both the final feature vector dimension (20) and the number of classes to be recognized (5) were low, the classification accuracy obtained using the synthesized features was only occasionally better than the classification accuracy obtained directly with the primitive features.

In [7], a similar CGP approach is applied for image classification and retrieval. The original 40-D feature vectors are synthesized into 10-D feature vectors. The results were compared in terms of classification accuracy against 10-D feature vectors obtained using multiple discriminant analysis (MDA) and also against a support vector machine (SVM) classifier using the original 40-D feature vectors. The databases used for testing consisted of 1200–6600 images from 12 to 50 classes. In all cases, the classification results obtained using the features synthesized by CGP were superior compared to features produced by MDA. Compared to the SVM classifier, the results were similar or better in a case where the database classes

consisted of multiple clusters in the original feature space. Also, the retrieval performance was compared using CGP and MDA generated features, and the CGP features were observed to yield better retrieval results than MDA features.

In [20, 21], the CGP-based features synthesis method and the expectation-maximization (EM) algorithm are combined into co-evolutionary feature synthesized expectation–maximization (CFS-EM). The main idea is to first use a minor part of the training data to reduce the feature space dimensionality and simultaneously to learn an initial Bayesian classifier using the CGP-based feature synthesis method and then to refine the classifier by applying the EM algorithm on the whole training data (the rest of which may be unlabeled) synthesized into the lower dimensionality. The classification and retrieval results obtained by CFS-EM were both improved compared to the CGP-only approach.

In our prior work [14], we introduced a new evolutionary feature synthesis method using multi-dimensional particle swarm optimization (MD PSO) [13, 17]. MD PSO is an extension of the well-known particle swarm optimization (PSO) [11], and it basically reforms the native structure of swarm particles in such a way that they can make inter-dimensional jumps with a dedicated dimensional PSO process. Therefore, in a multi-dimensional search space where the optimal solution dimensionality is also unknown, swarm particles can then seek for both positional and dimensional optima. In [14], new features are synthesized by applying a set of operators (mostly arithmetic) on a set of selected original low-level feature vector elements. The MD PSO particles are encoded so that the feature vector elements and operators are selected through the positional PSO process and the dimensional search in MD PSO allows the dimensionality of the new feature vectors to be optimized simultaneously. Compared to the original low-level features, this method could improve the retrieval performance measured using average normalized modified retrieval rank (ANMRR) [24] on an image database with 1000 images uniformly categorized into 10 classes about 0.14–0.25 depending on the original features. Over the same database, the classification accuracy was improved about 7–14 % units. The same method was also applied on audio classification and retrieval in [23] and it provided up to 15–25 % improvement in the retrieval performance. However, the performance of this method over larger image databases is quite modest due to the fact that only a single feature synthesis process has been used to synthesize features that discriminate all items in the database. This is indeed a serious drawback also for other standalone feature synthesizers proposed in the literature because it will be increasingly harder, if feasible at all, to find out a single set of transformations which can discriminate the items of all classes from each other when the number of classes is beyond a certain magnitude.

In a broader sense, besides the methods specially developed for feature synthesis, also well-known classifiers such as artificial neural networks (ANNs) and SVMs can be thought as a special kind of feature synthesizers. Commonly, ANNs used as classifiers take the original feature vectors as inputs and use 1-of-C output encoding (where C is the number of classes). Their output in an optimal case is a vector corresponding to the image class (e.g., the target vector $[1, 0, ..., 0]$ corresponds to first class and so on). Thus, ANNs try to learn a specific feature synthesizer that transforms each feature vector in a certain class to one corner of the C-dimensional cube. The simplest ANNs, single-layer perceptrons (SLPs), synthesize the original input features by forming in each neuron a weighted sum of all the input vector elements and passing it through a bounded nonlinear function (e.g., tangent hyperbolic or sigmoid) to give one of the output vector elements. Usually, only the weights of each input feature (and a bias) are optimized via the training algorithm used (e.g., back-propagation), while otherwise the synthesis follows a fixed path. In multi-layer perceptrons (MLPs), the network architecture is more complicated and must be set by the user or optimized with an algorithm capable of doing this simultaneously with weight optimization (e.g., MD PSO), but the principal approach remains the same than with SLPs.

SVMs, on the other hand, with a proper choice of the kernel used, can transform the original features of a two-class problem where the features are not linearly separable into new features in a higher dimension where linear separation is possible. The major drawback is the critical choice of the (non)linear kernel function along with its intrinsic parameters that may not be a proper choice for the problem at hand. Consider for instance, two sample feature synthesizers (FS-1 and FS-2) illustrated in Fig. 1, where for illustration purposes features are only shown in 1-D and 2-D, and only two-class problems are considered. In the case of FS-1, SVM with a polynomial kernel in quadratic form can make the proper transformation into 3-D so that the new (synthesized) features are linearly separable. However, for FS-2, a sinusoid with a proper frequency, f, should be used instead for a better class discrimination.

Also, several state-of-the-art techniques in computer vision and image categorization can be considered as feature synthesis methods. Different methods to find compact codes to represent images, such as classemes [26], PiCoDes [2] or semantic hashing [10, 19], generally start with some low-level features and after a training process a different set of features is obtained. Similarly, mid-level feature learning [25] transforms low-level features into semantically more meaningful features via a training process.

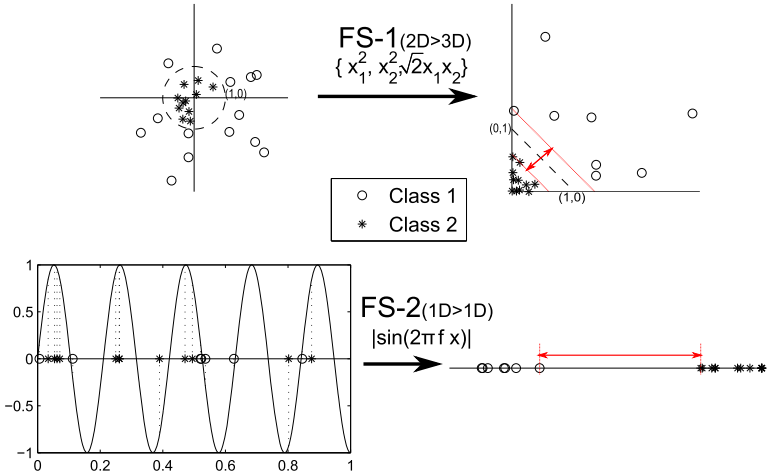


Fig. 1 Two sample feature synthesis performed on 2-D (FS-1) and 1-D (FS-2) feature spaces

Generally, these methods are not aiming to improve arbitrary low-level features, but on the contrary, their success is dependent on carefully selected low-level features. They may require manual selection of training categories [26], or their training may require a large number of training samples and a computationally expensive training process [25]. In both cases, the idea is to do training with a separate training data and then apply on similar general-purpose datasets. Compact binary codes provide a way to efficiently store millions of images in memory and to quickly find similar images in huge datasets where traditional image representations and search methods are impractical or impossible to use. They aim at preserving the discrimination power of the original features while transforming it into a form more suitable for large-scale image datasets. Thus, it is hardly feasible to apply any of these methods for feature synthesis over arbitrary low-level features extracted from any dataset especially if the dataset is highly specialized (e.g., different insects photographed in laboratory conditions). Such datasets will likely require retraining with manual selection and tuning operations because more specialized learning is needed to discriminate among similar classes.

3 Feature synthesis via one-against-all perceptrons

3.1 Motivation and objectives

As mentioned earlier, the motivation behind feature synthesis is to maximize the discrimination power of low-level features so that image classification and/or retrieval performance can be improved. Figure 2 illustrates an ideal feature synthesis operation where 2-D features of a 3-class database are successfully synthesized in such a way that the discrimination ability of the features clearly increases.

The main problem with the existing feature synthesis methods based on genetic programming is their computational complexity that restricts their practical usability in larger image databases. Also, a large number of image classes quickly forms an obstacle for a wider use of these methods even if enormous computational resources are available. To find a proper feature transformation that can simultaneously discriminate even a couple of dozen classes, is already such a complex task that any feature synthesis method will face severe problems in trying. Furthermore, most existing methods are quite dependent on several intrinsic parameters, which must be set manually and in advance. Our objective is to address all the above-mentioned deficiencies simultaneously. We want to create a feature synthesis method that can be easily (without parameter tuning or kernel selection) and relatively quickly applied on any kind of image database and with any low-level features extracted from the database.

The main realization behind the proposed feature synthesis method is that no matter how advanced the feature synthesis technique might be, the task of finding a transformation discriminating all the classes simultaneously will become too hard or even impossible when the number of classes grows sufficiently large. To remedy this drawback, we propose to use a synthesizer that is designed to discriminate one class from the rest at a time. We shall demonstrate that such a two-class problem is simple enough that ordinary perceptrons can handle it equally well or even better than more advanced evolutionary methods requiring significantly more computational resources. This will, in turn, allow the proposed feature synthesis method to better scale up to the larger databases.

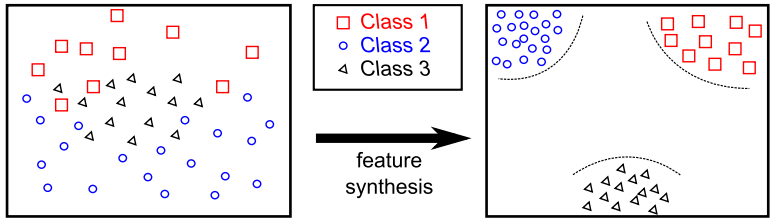


Fig. 2 An illustrative feature synthesis, which is applied to 2-D feature vectors of a 3-class database

3.2 Proposed feature synthesis method based on parallel one-against-all perceptrons

We propose a feature synthesis method based on parallel one-against-all perceptrons. Each perceptron is trained to discriminate a single class from the rest. We use original feature vectors directly as input vectors, and during the training, only the features of images belonging to that particular class are considered positive samples and features from the rest of the images negative. As usual with perceptrons, we use an output neuron per class i.e., each perceptron has two output neurons. We use [1 −1] as the target vector for positive items and [−1 1] for negative items. Finally, a new synthesized feature vector for a particular image is formed by giving its original feature vector as an input to each of the trained one-against-all perceptrons and concatenating all the outputs. Thus, the length of the new feature vector is twice the number of classes.

The whole feature synthesis process is illustrated in Fig. 3 for a limited sample case with three image classes and the original feature vector dimensionality being 5. It is necessary to train a one-against-all perceptron per each class i.e., three perceptrons are required. For the first perceptron, only the training samples from class 1 are given as positive samples (target output vector is set to [1 −1]), while the target output vector for the other training samples is set to [−1 1]. Similarly, for the second perceptron, only items from class 2 are considered positive samples, and for the third perceptron, only items from class 3. Finally, after training all the perceptrons, each original feature vector (belonging to both train and test sets) is given as an input to all the three perceptrons and the outputs are concatenated to form a new 6-dimensional feature vector for that item. In a very clear case, all the values in the new feature vector would be either 1s or −1s, but in reality, the values are usually somewhere between these two extremes. This might be also a desired property as it preserves some intra-class variations, and in CBIR, a user might want to get the most similar database items within a class first. Further note that, while in the figure, all the original feature vectors are illustrated with different symbols, the class information is only necessary for the training samples. Thus, the proposed method can also be used to synthesize more

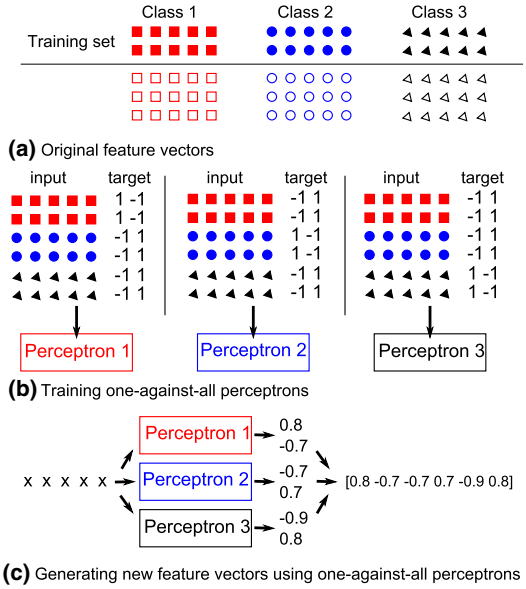


Fig. 3 Feature synthesis via parallel one-against-all perceptrons for a simplified 3-class database with 5-D original feature vectors

discriminative features even when only a minor part of the database has been labeled. Also, it is possible to easily replace perceptrons in the proposed approach with another one-against-all classifier with a similar output format.

The chosen approach with one-against-all perceptrons is easier and faster to train than all-at-once approaches due to its "divide and conquer" nature. Furthermore, it may be possible to train the proposed system incrementally. If a new class is added to the dataset, a new perceptron will be trained for the new class. The previously trained perceptrons need to be retrained only if they fail to classify the new samples with a required accuracy. Also, one-against-one classifiers could be used for similar feature synthesis; however, while the one-against-all approach will create new feature vectors, whose length is $2n$ for an n-class dataset, the feature vector length with one-against-one classifiers would be $n(n-1)$, which is significantly larger and hence may be non-desirable or even infeasible for certain applications.

The proposed feature synthesis method can successfully increase the discrimination power of even the most basic low-level features used in CBIR as will be demonstrated in Sect. 4. Naturally, the original features should have a certain description power. If (some elements of) the original features for a certain class are either isolated or if they contain some distinctive patterns, perceptrons can learn this and synthesize new features with superior discrimination power. On the other hand, if the original features totally lack description power, e.g., features created by a random number generator, there exists no distinct pattern that could be learned, and therefore, a successful feature synthesis is not possible. Moreover, if two images from different classes have a similar color structure while having a distinct texture or shape features, it is obvious that it is not possible to find a successful feature synthesizer for color features because they are not descriptive at all. Therefore, it is advantageous to have several different low-level feature types and to train a designated synthesizer for each feature type to maximize the overall benefit of the feature synthesis process.

The proposed perceptron-based feature synthesis (PFS) can be performed in several runs by using the synthesized feature vectors from the previous run as the input feature vectors for the next. This is illustrated in Fig. 4. The number of runs, R, can be specified in advance or adaptively determined, i.e., runs are carried out until the point where the fitness improvement is no longer significant.

3.3 Training and evolution methods

The proposed PFS can be used with any perceptron architecture and training method. To clarify the concepts used when introducing the experimental results, we will here briefly discuss the training and evolution algorithms applied, namely back-propagation (BP) and MD PSO. BP is the most well-known ANN training technique, which can be applied to a specific ANN architecture where the number of layers and the number of neurons in each layer has to be fixed a priori. BP sets the weights of each neuron exploiting gradient descent optimization. Like any gradient descent optimization method, BP is susceptible to getting trapped into local minima, and therefore, its performance is dependent on the initialization. Also, the selected network architecture may critically affect the final network performance.

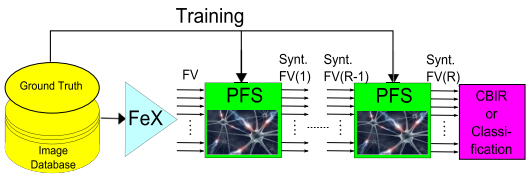


Fig. 4 Block diagram of Feature eXtraction (FeX) and proposed perceptron-based feature synthesis (PFS) method with consecutive *R* runs

MD PSO [13, 17] is a multi-dimensional (MD) extension of the basic PSO method [11]. In MD PSO, locations of the particles are candidate solutions similar to PSO, but unlike in basic PSO, MD PSO particles can search for solutions with different dimensionalities within a given dimensionality range, $\{D_{\min}, ..., D_{\max}\}$. In order to accomplish this, each particle has two sets of components, each of which has been subjected to one of the two independent and consecutive processes. The first one is the regular positional PSO, which takes place in the particles' current dimensionalities and the second one is a dimensional PSO, which allows particles to move between dimensionalities. The positional PSO comprises the traditional velocity updates and due positional shifts in the *N*-dimensional search (solution) space. Accordingly, for each dimensionality, the swarm keeps track of the best global position so far achieved (*gbest*) and each particle keeps track of its last position, velocity and personal best position in that particular dimensionality so that when it revisits the same dimensionality at a later time, it can perform the regular positional updates using this information. The dimensional PSO process may then move the particle to other dimensionalities where it will remember its positional status and will be updated through the positional PSO process, and so on. Similarly to the positional PSO, the dimensional PSO process uses the personal best dimensionality (in which the personal best fitness score has so far been achieved) of each particle and the global best dimensionality (*dbest*) to attract the particles toward a better dimensional solution. Finally, the *gbest* solution with dimensionality *dbest* represents the optimal solution and dimensionality, respectively. As the termination criterion for the MD PSO process we simply use a specified number of iterations.

When evolving ANNs with MD PSO, the positional PSO process can be used to optimize the neuron weights of a single ANN architecture, while the dimensional PSO process can simultaneously search for the optimal ANN architecture. The dimensionality range is replaced by an architecture space defined by the minimum and maximum number of hidden layers and the minimum and maximum number of neurons in each hidden layer (the number of neurons in input and output layers will be always dictated by the classification problem at hand). All the architectures within the architecture space will be enumerated and MD PSO can move between different architectures via the dimensional PSO process. The applied fitness function is simply the training mean squared error between the output of the corresponding MLP and the target output. A more

detailed description of using MD PSO for evolving MLPs can be found in [12].

When training with BP, to compensate for MD PSO's ability to search for the optimal network architecture simultaneously with the training of network weights, we will train separately each MLP configuration in the architecture space and use the best performance obtained by the optimal architecture found as the final performance. We will call this sequential search method exhaustive BP training over the architecture space.

4 Experimental results

4.1 Databases, features, and test settings

We used MUVIS framework (http://muvis.cs.tut.fi/) to create and index three different databases that are used in the experiments in this section. The databases are:

1. *C_10* image database: There are 1000 medium resolution (384 × 256 pixels) images obtained from Corel repository covering 10 diverse classes: 1—Natives, 2—Beach, 3—Architecture, 4—Bus, 5—Dino Art, 6—Elephant, 7—Flower, 8—Horse, 9—Mountain, and 10—Food. Each class has 100 images.
2. *CC_30* image database: There are 4245 images from 30 diverse classes that are obtained from both Corel and Caltech [8] image repositories. The number of images in a class varies from 80 to 435.
3. *F_20* image database: There are 34481 images from 20 diverse classes obtained by selecting the first 20 classes from Flickr database [5] and removing all the images with multiple tags. The number of images in database classes is heavily imbalanced varying from 12 to 17252.

The features extracted from the databases along with the parameters used are given in Table 1. It should be noted that the choice of the original features and their parameters is irrelevant to the proposed feature synthesis approach due its generic nature and the details are given here only for reproducibility of the experiments. We chose a set of basic low-level features with varying (but limited) discrimination abilities and CBIR performance. Neither manual feature selection nor parameter tuning is applied to boost the performance. As a preprocessing step, all the features were normalized between 0 and 1.

In most of the experiments (Sects. 4.2, 4.3), we used MD PSO for evolving MLPs. In these sections, we used an architecture space with the maximum of one hidden layer and number of neurons in this layer set to 4–10. The MD PSO parameters were set as follows: The swarm size, S, was set to 200 and the number of iterations to 2000. The positional search space range, $\{X_{\min}, ..., X_{\max}\}$, was set to $\{-2, ..., 2\}$ and positional velocity range, $\{V_{\min}, ..., V_{\max}\}$, to $\{-X_{\max}/10, ..., X_{\max}/10\}$. The dimension range, $\{D_{\min},...,D_{\max}\}$, was set according to the architecture space and the dimensional velocity range $\{VD_{\min}, ..., VD_{\max}\}$, to $\{-5, ..., 5\}$. See [12] for a detailed description of the meaning of these parameters in evolving MLPs with MD PSO.

The maximum number of PFS runs (as illustrated in Fig. 4) is set to 20 for *C_10* and *CC_30* databases and to three for *F_20* database. All the feature synthesis results are presented by computing the average of the classification or CBIR performances from 10 separate repetitions

In Sect. 4.2, PFS is applied for image classification and in Sect. 4.3 for CBIR. Besides comparing the results against those obtained by using the original features, comparisons against state-of-the-art methods are also conducted. Section 4.4 evaluates the significance of the selected training method and architecture space for PFS.

4.2 Feature synthesis experiments for classification

First, we evaluated the discrimination ability of the original features by using 45 % of the images to train a nearest centroid classifier and computing classification error among the remaining 55 % of the images. In the feature synthesis experiments, we used the same 45 % of images to train the synthesizer first and then used the synthesized features of these same images to train the nearest centroid classifier. The overall classification performance was computed by normalizing the centroid distances of the

Table 1 Features extracted from databases and parameters used

Feature	Parameters	Dim.
HSV color histogram (HSV)	$H = 6, S = 2, V = 2, SimTh = 30$	24
Dominant color histogram(DOCQ)	$N^{DC}_{\max} = 6, SimTh = 15, A_D = 0.02$	27
Color structure descriptor (CSD)	$Bins = 32$	32
Local binary pattern (LBP)	$A = 3, qTh = 5$	16
Gabor	$Scale = 4, Orient = 6$	48
Ordinal co-occurence (ORDC)	$d = 3, o = 4$	36
Edge histogram Dir. (EHD)	$Bsiz = 4, DC = 11$	5

individual features (the maximum distance between an item and the centroid was set to one for each feature) and setting the final class according to the combined distance. The nearest centroid classification results in terms of the average test classification error are given in Table 2.

For *F_20* database, evaluating classification performance in terms of classification error is not enough as the class sizes are so imbalanced that simply classifying all the items into one or two classes would also significantly improve the classification performance. Therefore, we used also GPR, the geometric mean of true positive rate (recall) and precision rate (precision). In [6], GPR has been found to effectively compromise between the success in the minority class and the error in the majority class in a two-class classification problem. In this work, we computed the GPR values on the test dataset individually for each class and took the average (AGPR) as the final classification measure. The nearest centroid classification results in terms of AGRP for *F_20* database are given in Table 3.

The results clearly indicate a crucial improvement in the classification accuracy, leading to a conclusion that the proposed feature synthesis method can indeed synthesize such features that can be classified more efficiently and accurately. The only exception is that PFS fails to improve the classification performance of the EHD feature for *F_20* database. On *C_10* database, the best possible classification performance has practically been achieved already at the first run (only minimal improvement on LBP and ORDC features at later iterations), but with the larger *CC_30* database, the latter runs have been more beneficial. On *F_20* database, the improvement at later iterations is generally even more significant. For the ORDC feature, the results at the first iteration are worse than the original results, but a significant improvement at later iterations has still been obtained. It should be also noted that with *F_20* database only three consecutive PFS runs ($R = 3$) were performed due to the large size of the database and it is highly probable that further improvement could be obtained with more runs.

To better evaluate the obtained discrimination power, we compare our classification results against some of the most powerful generally known classifiers: MLPs, SVMs and RF. For MLPs, we apply exhaustive BP training over the architecture space defined above. For BP training, the learning rate parameter, η, was set to 0.02 and the number of iterations to 5000. For SVMs, we employed the libSVM library [4] using the *one-against-one* topology. We used all standard kernel types (linear, polynomial, radial basis function (RBF) and sigmoid) and for each kernel type, we applied a similar exhaustive search to find out the best parameters, i.e., we trained with all the combinations of the penalty parameter $C = 2^n$ for $n = 0, \ldots, 4$ and parameter $\gamma = 2^{-n}$ for $n = 0, \ldots, 4$ and finally used the results obtained by the best parameter combination. For RF, we similarly selected the best number of trees within the forest, searching from 10 to 50 in steps of 10. For all classifiers, 45 % of the items were used for training. The results in terms of the classification error are given in Table 4. For *F_20* database, the results are also given in terms of AGPR in Table 5.

The results on *C_10* and *CC_30* databases show that the proposed feature synthesis method can improve the discrimination power of the low-level features to such level that using the synthesized features even the simple nearest centroid classifier can obtain results comparable to best results

Table 2 Test classification errors of the nearest neighbor classifier using original and synthesized features

	CSD	DOCQ	EHD	HSV	LBP	ORDC	Gabor	All
C_10								
Orig.[a]	31.82	50.36	45.09	44.55	62.36	54.36	43.27	20.91
PFS1[b]	**25.29**	43.56	**38.31**	**33.04**	**35.87**	32.18	**29.67**	**14.55**
PFS[c]	**25.29**	**43.45**	**38.31**	**33.04**	**35.87**	**31.78**	**29.67**	**14.55**
C_30								
Orig.[a]	62.39	77.50	77.80	71.00	84.72	77.84	65.28	51.72
PFS1[b]	**46.93**	61.05	58.40	60.16	55.40	51.13	41.02	30.69
PFS[c]	**46.93**	**60.53**	**56.78**	**57.47**	**52.48**	**49.80**	**40.86**	**29.99**
F_20								
Orig.[a]	75.64	97.31	**85.65**	91.68	77.03	81.30	77.73	73.57
PFS1[b]	46.22	67.09	87.05	53.66	75.02	87.29	82.32	57.02
PFS[c]	**43.90**	**55.55**	86.25	**51.02**	**68.60**	**44.95**	**50.28**	**44.21**

Best values are in bold

[a] Original features

[b] Features synthesized by one iteration of PFS

[c] Best features synthesized by PFS

Table 3 Average geometric mean of true positive rate and precision rate on testing data for a nearest centroid classifier using original and synthesized features

F_20	CSD	DOCQ	EHD	HSV	LBP	ORDC	Gabor	All
Orig.[a]	0.105	0.039	**0.081**	0.059	0.076	0.080	0.089	0.128
PFS1[b]	0.131	0.077	0.075	0.100	0.096	0.076	0.098	0.151
PFS[c]	**0.133**	**0.082**	0.080	**0.104**	**0.105**	**0.126**	**0.139**	**0.171**

Best values are in bold

[a] Original features

[b] Features synthesized by one iteration of PFS

[c] Best features synthesized by PFS

Table 4 Test classification errors of a nearest centroid classifier using synthesized features and state-of-the-art classifiers using original features

	CSD	DOCQ	EHD	HSV	LBP	ORDC	Gabor	Aver.
C_10								
PFS+NCC[a]	25.29	43.45	38.30	33.04	**35.87**	31.78	29.67	**33.92**
SVM (lin.)	24.18	40.73	38.18	33.09	55.09	39.82	29.45	37.22
SVM (polyn.)	25.64	41.09	43.64	61.82	62.73	**30.55**	28.36	41.97
SVM (RBF)	**22.91**	39.45	**37.64**	**30.73**	53.45	36.55	**26.36**	35.30
SVM (sigm.)	30.00	55.27	54.36	50.55	66.73	58.91	49.82	52.23
RF	30.36	**38.00**	48.91	31.82	40.73	44.36	36.55	38.68
MLP, BP	27.95	48.80	41.93	34.45	40.22	35.22	31.55	37.16
C_30								
PFS+NCC[a]	46.93	60.53	56.78	57.47	**52.48**	49.80	40.86	**52.12**
SVM (lin.)	44.14	61.88	59.74	58.29	68.61	53.56	40.68	55.27
SVM (polyn.)	42.99	**54.37**	69.13	73.65	70.41	**46.91**	38.42	56.55
SVM (RBF)	**40.43**	56.55	57.10	**57.14**	67.29	49.30	**37.61**	52.20
SVM (sigm.)	57.48	81.11	80.43	76.67	81.30	81.11	71.81	75.70
RF	49.85	55.82	60.77	61.11	55.91	55.69	47.63	55.25
MLP, BP	52.64	61.66	**55.03**	57.44	53.11	50.74	44.89	53.64
F_20								
PFS+NCC[a]	43.90	55.55	86.25	51.02	68.60	44.95	50.28	57.22
SVM (lin.)	38.87	44.53	35.58	41.68	35.35	33.07	33.38	37.49
SVM (RBF)	**35.36**	**41.57**	**35.46**	**39.52**	**34.78**	**31.67**	**32.13**	**35.78**

Best values are in bold

[a] Nearest centroid classifier using features synthesized by the proposed method

Table 5 Average geometric mean of true positive rate and precision rate of a nearest centroid classifier using synthesized features and state-of-the-art classifiers using original features on testing data

F_20	CSD	DOCQ	EHD	HSV	LBP	ORDC	Gabor	Aver.
PFS+NCC[a]	**0.133**	**0.082**	0.080	**0.104**	**0.105**	**0.126**	**0.139**	**0.110**
SVM (lin.)	0.077	0.058	0.086	0.063	0.082	0.094	0.094	0.079
SVM (RBF)	0.118	0.063	**0.087**	0.066	0.086	0.096	0.097	0.087

Best values are in bold

[a] Nearest centroid classifier using features synthesized by the proposed method

obtained with several state-of-the-art classifiers trained using the best possible parameters. Furthermore, the proposed method usually generates the near top classification results for every feature, while for the other classifiers performance level varies significantly among different features. This is demonstrated also by the fact that the best average classification result is obtained by the proposed method. For *F_20* database, the classification errors obtained

by both applied SVM configurations are clearly lower than those obtained by PFS. However, further examination of the confusion matrices shows that this superiority can be fully explained by the fact that SVMs classify all the items into 2–3 classes. The proposed method can, on the other hand, synthesize from all the low-level features (except EHD), such features that the classification performance can be improved also on minority classes as evident from Table 5. In other words, the discrimination power of the features can be improved also in such an imbalanced case.

4.3 Feature synthesis experiments for content-based image retrieval

Our second goal was to synthesize such feature vectors that their performance in CBIR is superior to the original low-level features. We evaluate the CBIR performance in terms of ANMRR and average precision (AP) computed by the batch query, i.e., querying all images in the database. For each feature (both low-level and synthesized), Euclidean distance measure between feature vectors is used to com-

Table 6 Average normalized modified retrieval rank using original and synthesized features

	CSD	DOCQ	EHD	HSV	LBP	ORDC	Gabor	All
C_10								
Orig.[a]	50.12	73.17	61.68	61.93	68.62	62.28	56.13	43.86
PFS1[b]	**24.73**	49.08	47.20	32.47	46.30	42.13	34.61	18.15
PFS[c]	**24.73**	**42.51**	**41.14**	**32.40**	**36.80**	**34.08**	**31.24**	**17.14**
C_30								
Orig.[a]	67.84	81.84	76.38	78.31	80.67	75.98	70.71	60.39
PFS1[b]	47.42	72.35	67.92	63.86	67.33	63.88	50.82	41.60
PFS[c]	**44.51**	**63.48**	**63.49**	**59.22**	**56.40**	**54.51**	**42.80**	**33.49**
F_20								
Orig.[a]	71.21	75.74	70.12	74.96	70.89	71.91	72.57	69.91
PFS1[b]	66.23	72.66	66.17	69.63	65.17	64.42	64.79	61.52
PFS[c]	**65.99**	**72.29**	**66.06**	**69.18**	**64.55**	**63.40**	**64.08**	**60.21**

Best values are in bold

[a] Original features

[b] Features synthesized by one iteration of PFS

[c] Best features synthesized by PFS

Table 7 Average precision using original and synthesized features

	CSD	DOCQ	EHD	HSV	LBP	ORDC	Gabor	All
C_10								
Orig.[a]	47.70	25.20	36.49	35.89	29.93	36.20	41.73	53.47
PFS1[b]	**73.35**	49.17	51.41	65.51	52.36	56.24	63.71	79.94
PFS[c]	**73.35**	**55.57**	**57.53**	**65.56**	**61.83**	**64.31**	**66.93**	**80.86**
CC_30								
Orig.[a]	30.40	17.03	22.34	20.85	18.11	22.50	27.55	37.30
PFS1[b]	50.35	26.14	30.82	34.59	30.99	34.37	47.14	56.00
PFS[c]	**53.90**	**35.30**	**35.48**	**39.62**	**44.42**	**41.40**	**55.96**	**63.17**
F_20								
Orig.[a]	28.45	24.12	29.41	24.73	28.66	27.49	27.00	29.58
PFS1[b]	33.50	27.21	33.59	30.10	34.60	35.30	34.91	38.00
PFS[c]	**33.77**	**27.61**	**33.72**	**39.62**	**35.28**	**36.47**	**35.72**	**39.31**

Best values are in bold

[a] Original features

[b] Features synthesized by one iteration of PFS

[c] Best features synthesized by PFS

pute the (dis-)similarity distance. In case of the overall retrieval performance, each feature was given an equal weight when evaluating the overall distance between two images. The training set size of 45 % was used, while the retrieval performance is given over the whole database. The results in terms of ANMRR are given in Table 6 and in terms of AP in Table 7.

Table 8 Average normalized modified retrieval rank and average precision using features synthesized by the proposed method and features produced by Collective Network of Binary Classifier Framework

	C_10 ANMRR	AP	*CC_30* ANMRR	AP
PFS	**17.14**	**80.86**	**33.49**	**63.17**
CNBC (MD PSO)	31.09	65.01	43.04	54.47
CNBC (BP)	23.86	74.26	46.44	52.21

Best values are in bold

The results indicate that a significant improvement in retrieval performance has been obtained. Compared to the test classification errors on *C_10* and *CC_30* databases given in Table 2, a higher level of improvement can now be achieved during the latter iterations of the proposed feature synthesis technique. This leads to the assumption that most of this improvement is within the training set, while the performance on the test set (majority of the database) does not significantly change. On *F_20* database, similar behavior cannot be observed most likely because only three PFS runs were performed and also the classification performance on the test set (Tables 2, 3) was still significantly improving.

Table 8 shows comparison to Collective Network of Binary Classifier Framework (CNBC) [16, 17], which has been shown to produce competitive CBIR results using low-level features [15]. The same databases, low-level features and training sets were used to evolve/train the CNBC by both techniques: MD PSO and exhaustive BP.

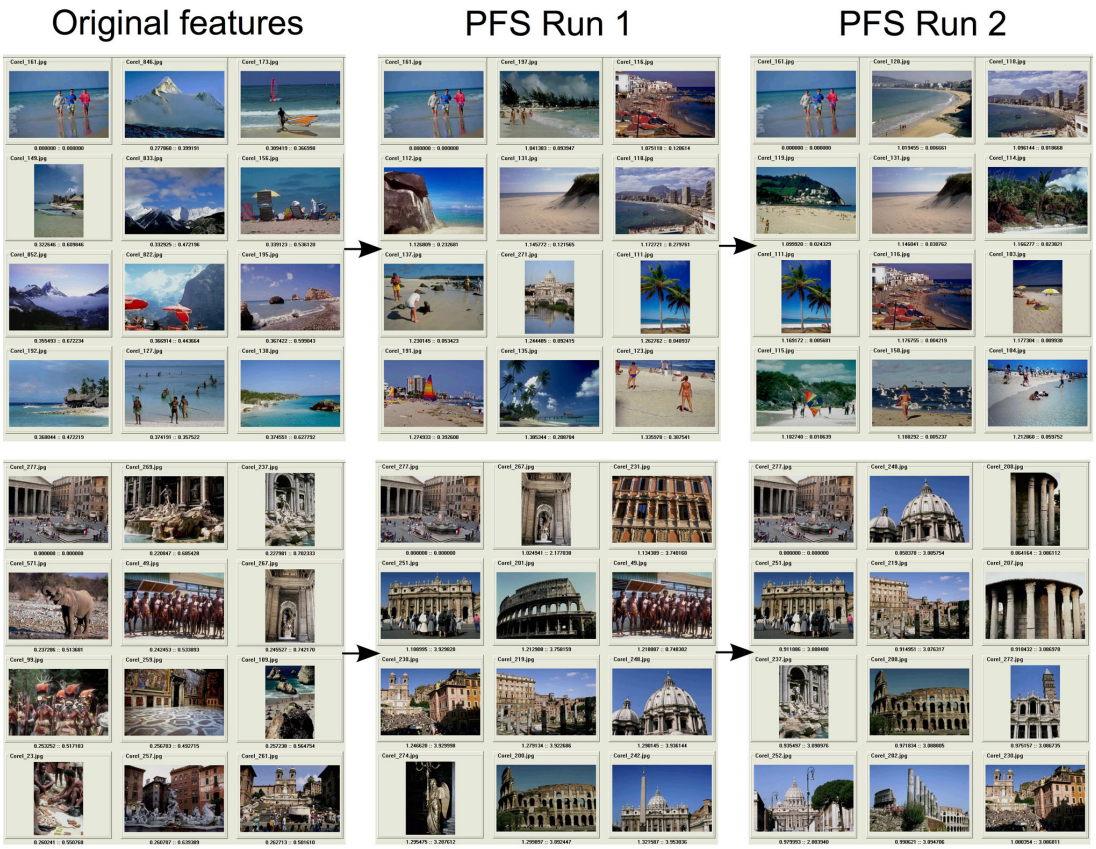


Fig. 5 Two sample queries on *C_10* database using original (*left*) and synthesized features with single (*middle*) and two (*right*) runs. *Top-left* is the query image

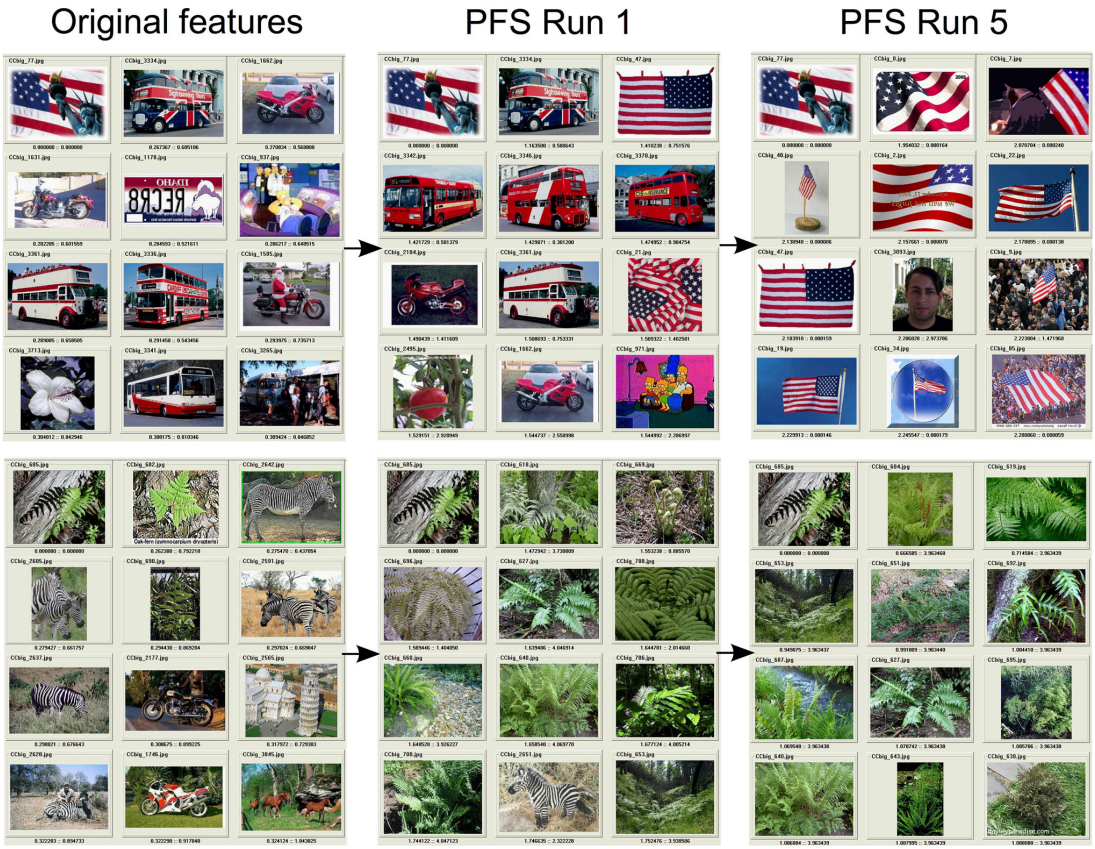


Fig. 6 Two sample queries on *CC_30* database using original (*left*) and synthesized features with single (*middle*) and five (*right*) runs. *Top-left* is the query image

Based on the overall retrieval performances given in Table 8, it is clear that the proposed method can achieve a superior retrieval performance on both databases.

Figure 5 illustrates two sample queries on *C_10* database where neither of the selected query images is not in the training set. Both queries are performed using the original features, features synthesized by a single run and by two runs of the proposed feature synthesis method. It is obvious that the synthesized features have a superior discrimination power than the original features, and in turn, a better retrieval performance can be achieved. Most of the improvement generally takes place at the first run, but further runs help to fine-tune the result. Figure 6 similarly shows two sample queries on *CC_30* database with the exception that the last query snapshots are taken after 5 PFS runs.

4.4 Comparative evaluations on training methods and architecture spaces

To evaluate the significance of the training method and the architecture space used, we repeated the earlier classification and retrieval experiments using different architecture spaces and either MD PSO evolution or exhaustive BP training. Besides the changing architecture space, we used MD PSO with the same parameters defined in Sect. 4.1. For BP, the learning rate parameter, η, was set to 0.02 and the number of iterations to 5000. The smallest architecture space consisted of only a SLP, the medium architecture space had the maximum of one hidden layer and number of neurons in this layer set to 4–10, and the large architecture space had the maximum of two hidden layers with 4–8 neurons in the first hidden layer and 8–16 neurons in the

Table 9 Test classification errors for a nearest neighbor classifier using features synthesized with different training methods and architecture spaces

C_10	MD PSO Small	MD PSO Medium	MD PSO Large	BP Small	BP Medium	BP Large
Run 1	14.85	14.55	15.13	**14.53**	16.00	16.42
Best run	14.62	14.55	15.13	**14.47**	16.00	16.42

CC_30	MD PSO Small	MD PSO Medium	BP Small	BP Medium
Run 1	32.42	**30.69**	33.67	31.66
Best run	30.21	**29.99**	33.49	31.66

Best values are in bold

Table 10 Average normalized modified retrieval rank for features synthesized with different training methods and architecture spaces

C_10	MD PSO Small	MD PSO Medium	MD PSO Large	BP Small	BP Medium	BP Large
Run 1	23.62	18.15	19.03	24.77	18.02	**17.69**
Best run	17.26	**17.14**	17.39	18.20	17.95	17.69

CC_30	MD PSO Small	MD PSO Medium	BP Small	BP Medium
Run 1	50.09	41.60	51.02	**39.32**
Best run	42.60	**33.49**	50.95	34.13

Best values are in bold

second. As similar results were observed with all the features used, only the results using all the features listed in Table 1 are given here. Classification results are given in Table 9 and retrieval results in Table 10.

Generally, the difference between the results obtained by MD PSO and exhaustive BP training methods is small. The main benefit of using MD PSO seems to be its better capability to improve the synthesis on latter runs, especially on *CC_30* database. For *C_10* database, also the differences between different architectures in terms of classification and retrieval performances are modest. SLPs achieve the lowest test classification error, but their initial retrieval performance is worse than the performance obtained with medium and large architecture spaces. This suggests that for the larger architecture spaces, some overlearning has already occurred at the first run, while with the simplest architecture, SLP, similar overlearning only takes place after a couple of iterations (until some point the overlearning on training set will improve the retrieval accuracy as the whole database including the training set is considered when the retrieval scores are computed). *CC_30* database, on the other hand, is so much more complex that SLPs cannot find best solutions anymore. With larger architectures, the test classification error more often keeps improving during several iterations i.e., overlearning on training set is no longer such a problem for them.

5 Conclusion

In this paper, we proposed a new feature synthesis technique, PFS, which transforms the original features using parallel one-against-all perceptrons. PFS can be easily applied on any low-level features and on any database, where some ground-truth data exists. The main advantage of the proposed method is its computational simplicity compared to the existing feature synthesis methods commonly based on evolutionary algorithms. Along with its simplicity, its one-against-all topology is essential when applied on larger databases. Finding such a feature synthesizer that can simultaneously discriminate all the database classes from each other may become a too hard or even an impossible problem when the number of classes grows. To overcome this problem, the proposed method trains a dedicated one-against-all perceptron for each class to efficiently discriminate that class from the rest. Only at the end of the process, these class-specific syntheses are concatenated into one composite feature vector. It is obvious that this is a

"divide and conquer" type of approach that divides a difficult, for larger databases probably infeasible, synthesis problem into several easier two-class synthesis problems. This kind of problem implementation also allows efficient exploiting of modern parallel processing facilities.

Experimental results over benchmark image databases demonstrate that the proposed system can significantly improve the discrimination power of the original features and thus achieve a crucial improvement in classification and CBIR performances. We can conclude that as long as some ground-truth information is available for a subset of an image database, any low-level features extracted from the database can be easily replaced with new features having a higher discrimination (and description) capability. This is appealing especially for existing applications where mid-level feature extraction is not a viable option and/or only a certain set of features is available, while the original data is either missing or incomplete.

In the future, we aim to apply the proposed feature synthesis method on significantly larger databases. While its computational simplicity and one-against-all topology support increasing database sizes, larger databases (~ 100 thousands images) will still require certain modifications. For instance, an efficient training sample selection should be applied to obtain the maximum performance with the minimum amount of training. Furthermore, some large image classes may be so diverse that also their discrimination from the other classes using a single synthesizer gets difficult. It can be possible to find clusters of similar items within such diverse classes and to train a separate one-against-all perceptron for each cluster. These are topics for our future research.

References

1. Bäck T, Schwefel HP (1993) An overview of evolutionary algorithms for parameter optimization. Evolut. Comput. 1(1):1–23. doi:10.1162/evco.1993.1.1.1
2. Bergamo A, Torresani L, Fitzgibbon A (2011) Picodes: learning a compact code for novel-category recognition. In: Shawe-Taylor J, Zemel R, Bartlett P, Pereira F, Weinberger K (eds) Advances in neural information processing systems 24, pp 2088–2096
3. Bhanu B, Yu J, Tan X, Lin Y (2004) Feature synthesis using genetic programming for face expression recognition. In: Proceedings of genetic and evolutionary computation conference, Part II, Seattle, pp 896–907. doi:10.1007/978-3-540-24855-2_103
4. Chang CC, Lin CJ (2011) Libsvm: a library for support vector machines. ACM Trans Intell Syst Technol 2(3):27:1–27:27. doi:10.1145/1961189.1961199
5. Chua TS, Tang J, Hong R, Li H, Luo Z, Zheng Y (2009) Nus-wide: a real-world web image database from national university of Singapore. In: Proceedings of the ACM international conference on image and video retrieval, ACM, New York, pp 48:1–48:9. doi:10.1145/1646396.1646452
6. Daskalaki S, Kopanas I, Avouris N (2006) Evaluation of classifiers for an uneven class distribution problem. Appl Artif Intel 20:1–37. doi:10.1080/08839510500313653
7. Dong A, Bhanu B, Lin Y (2005) Evolutionary feature synthesis for image databases. In: Proceedings of 7th IEEE workshop on application of computer vision, vol 1, pp 330–335. doi:10.1109ACVMOT.2005.50
8. Fei-Fei L, Fergus R, Perona P (2007) Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. Comput Vis Image Underst 106(1):59–70. doi:10.1016/j.cviu.2005.09.012
9. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning, 1st edn. Addison-Wesley Longman Publishing Co. Inc, Boston
10. Jiang YG, Wang J, Xue X, Chang SF (2013) Query-adaptive image search with hash codes. IEEE Trans Multimed 15(2):442–453. doi:10.1109/TMM.2012.2231061
11. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of the IEEE international conference on neural networks, vol 4, pp 1942–1948. doi:10.1109/ICNN.1995.488968
12. Kiranyaz S, Ince T, Yildirim A, Gabbouj M (2009) Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. Neural Netw 22(10):1448–1462. doi:10.1016/j.neunet.2009.05.013
13. Kiranyaz S, Ince T, Yildirim A, Gabbouj M (2010) Fractional particle swarm optimization in multidimensional search space. IEEE Trans Syst Man Cybern Part B Cybern 40(2):298–319. doi:10.1109/TSMCB.2009.2015054
14. Kiranyaz S, Pulkkinen J, Ince T, Gabbouj M (2011a) Multi-dimensional evolutionary feature synthesis for content-based image retrieval. In: Proceedings of 18th IEEE international conference on image processing, Brussels, pp 3645–3648. doi:10.1109/ICIP.2011.6116508
15. Kiranyaz S, Uhlmann S, Pulkkinen J, Gabbouj M, Ince T (2011b) Collective network of evolutionary binary classifiers for content-based image retrieval. In: Proceedings of IEEE workshop on evolving and adaptive intelligent systems, Paris, pp 147–154. doi:10.1109/EAIS.2011.5945925
16. Kiranyaz S, Ince T, Uhlmann S, Gabbouj M (2012) Collective network of binary classifier framework for polarimetric sar image classification: an evolutionary approach. IEEE Trans Syst Man Cybern Part B Cybern 42(4):1169–1186. doi:10.1109/TSMCB.2012.2187891
17. Kiranyaz S, Ince T, Gabbouj M (2014) Multidimensional particle swarm optimization for machine learning and pattern recognition. Springer, NewYork
18. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
19. Li P, Wang M, Cheng J, Xu C, Lu H (2013) Spectral hashing with semantically consistent graph for image indexing. IEEE Trans Multimed 15(1):141–152. doi:10.1109/TMM.2012.2199970
20. Li R, Bhanu B, Dong A (2005) Coevolutionary feature synthesized em algorithm for image retrieval. In: Proceedings of the 13th annual ACM international conference on multimedia, ACM, New York, pp 696–705. doi:10.1145/1101149.1101304
21. Li R, Bhanu B, Dong A (2008) Feature synthesized em algorithm for image retrieval. ACM Trans Multimed Comput Commun Appl 4(2):10:1–10:24. doi:10.1145/1352012.1352014
22. Lin Y, Bhanu B (2005) Evolutionary feature synthesis for object recognition. IEEE Trans Syst Man Cybern Part C Appl Rev 35(2):156–171. doi:10.1109/TSMCC.2004.841912
23. Mäkinen T, Kiranyaz S, Raitoharju J, Gabbouj M (2012) An evolutionary feature synthesis approach for content-based audio retrieval. EURASIP J Audio Speech Music Process 2012(23):1–23. doi:10.1186/1687-4722-2012-23
24. Manjunath B, Salembier P, Sikora T (2002) Introduction to MPEG-7. Wiley, San Francisco
25. Mittelman R, Lee H, Kuipers B, Savarese S (2013) Weakly supervised learning of mid-level features with beta-bernoulli

process restricted boltzmann machines. In: Proceedings of IEEE conference on computer vision and pattern recognition, pp 476–483. doi:10.1109/CVPR.2013.68
26. Torresani L, Szummer M, Fitzgibbon A (2010) Efficient object category recognition using classemes. In: Proceedings of 11th European conference on computer vision, Part I, Heraklion, pp 776–789
27. Yu J, Bhanu B (2006) Evolutionary feature synthesis for facial expression recognition. Pattern Recognit Lett 27(11):1289–1298. doi:10.1016/j.patrec.2005.07.026

Publication VII

Jenni Raitoharju, Serkan Kiranyaz, Moncef Gabbouj, "Training Radial Basis Function Neural Networks for Classification via Class-Specific Clustering," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 12, pp. 2458 – 2471, Dec. 2016.

Training Radial Basis Function Neural Networks for Classification via Class-specific Clustering

Jenni Raitoharju, *Member, IEEE*, Serkan Kiranyaz, *Senior, IEEE*, Moncef Gabbouj, *Fellow, IEEE*

***Abstract*—In training radial basis function neural networks, the locations of Gaussian neurons are commonly determined by clustering. Training inputs can be clustered either on a fully unsupervised manner (input clustering) or some supervision can be introduced, for example, by concatenating the input vectors with weighted output vectors (input-output clustering). In this paper, we propose to apply clustering separately for each class (class-specific clustering). The idea has been used in some previous works, but without evaluating the benefits of the approach. We compare the class-specific, input, and input-output clustering approaches in terms of classification performance and computational efficiency when training radial basis function neural networks. To accomplish this objective, we apply three different clustering algorithms and conduct experiments on 25 benchmark datasets. We show that the class-specific approach significantly reduces the overall complexity of the clustering and our experimental results demonstrate that it can also lead to a significant gain in the classification performance especially for the networks with relatively few Gaussian neurons. Among other applied clustering algorithms, we combine for the first time a dynamic evolutionary optimization method, multi-dimensional particle swarm optimization, and the class-specific clustering to optimize the number of cluster centroids and their locations.**

***Index Terms*—radial basis function networks, clustering methods, supervised learning, particle swarm optimization**

I. Introduction

RADIAL basis function neural networks (RBFNNs) have been successfully applied in several areas such as medical diagnostics [1], robotics [2], dynamic system design [3], and stock index forecasting [4]. RBFNNs are three-layered feedforward neural networks with universal approximation ability. The input layer is passive, the hidden radial basis function (RBF) neurons transform the input data to a new feature space using a strictly positive radially symmetric activation function, and the linear output layer supplies the network's response to the transformed input data. Gaussian basis functions are commonly used as activation functions. The output of the j^{th} output neuron can be given as

$$y_j = \sum_{k=1}^{K} w_{kj} g_k(\mathbf{x}) = \sum_{k=1}^{K} w_{kj} \ exp\left(\frac{-\left\|\mathbf{x}-\boldsymbol{\mu}_k\right\|^2}{2\sigma_k^2}\right), \quad (1)$$

where K is the number of Gaussian neurons, w_{kj} is the weight between the k^{th} Gaussian neuron and the j^{th} output neuron, $\boldsymbol{\mu}_k$ and σ_k are the center location and width of the k^{th} Gaussian neuron respectively, and $\mathbf{x}$ is an input vector.

The most significant step in the training of RBFNNs is the selection of the number of centroids and their locations. The selection is commonly performed by clustering the training items. The clustering can be performed on a fully unsupervised manner (input clustering) or some supervision can be incorporated into the clustering process. In input-output clustering the input vectors are concatenated with the outputs and the clustering is performed on these concatenated vectors. In this paper, we propose to perform clustering separately for each class. Some individual works [5], [6] have previously applied this approach. However, these works do not compare their class-specific approach with the other clustering approaches and the approach is still rarely used.We show that the class-specific clustering approach is faster than the traditional clustering approaches and our experimental results show that also significantly better classification results can be obtained using the class-specific approach especially when networks with lower number of RBF neurons are trained. We perform an extensive set of the experiments using three different clustering approaches. We apply the most well-known clustering algorithm, K-means, with three different cluster numbers. We use APC-III, which is a dynamic clustering method earlier used for the class-specific clustering in RBFNN training. We now apply the same algorithm also with the input and input-output clustering approaches. Finally, we carry out experiments using multi-dimensional particle swarm optimization (MD PSO) and fractional global best formation (FGBF). As a dynamic clustering method based on stochastic optimization, this method can avoid local optima and is not dependent on the starting point, but it is also computationally more expensive than the other considered algorithms. To our knowledge this is the first time of proposing such a combination of an evolutionary method and the class-specific clustering approach. Our main objective is to show that all these different clustering algorithms can benefit from the class-specific clustering approach. Furthermore, we want to clearly show the significance of the different clustering approaches and, therefore, we keep the rest of training process as simple as possible.

The rest of the paper is organized as follows. In Section II, the RBFNN training process is introduced in detail and recent advances on the topic are also introduced. The input, input-output, and class-specific clustering approaches are presented in detail in Section III. The clustering algorithms used in the experiments are introduced in Section IV and in Section V we introduce the experimental setup and give the results with comparative evaluations. Finally, Section VI concludes the paper and suggests topics for future work.

II. RBFNN Training

To train a RBFNN, one needs to decide the number of hidden neurons, assign the values of their center locations

and widths, and finally assign the weight values. It has been experimentally demonstrated that the RBF neuron center locations are more significant to the evaluation performance than their widths or the weights [7]. Also the number of hidden neurons impacts the network's performance significantly: too few neurons may be insufficient to learn a model, while too many neurons may lead to overfitting [8], [9].

The traditional RBFNN training strategies can be divided to one-phase, two-phase, and three-phase learning [10]. In one-phase learning only the weights between the Gaussian and output neurons are adjusted using a supervised optimization method. The center locations are set by randomly selecting a desired number of training items as RBF centers and the center widths are typically set to a fixed value. In two-phase learning, the center locations and widths are assigned in the first phase and the network weights are determined in the second phase. In the three-phase training the RBFNN is first initialized using two-phase learning and in the third phase the whole network is further adjusted using an optimization procedure, e.g. back-propagation learning [11]. The third training phase can generally improve the performance obtained by the first two phases [10]. However, in this article we will mainly concentrate on the two-phase learning.

Originally, in their regularization networks quite similar to RBFNNs, Poggio and Girosi placed the RBF centers at each training item [12]. However, this may lead to overfitting and to computationally expensive large networks when the number of training items is high. In [13], Poggio and Girosi suggested that a smaller number of centers should be used to overcome these problems. They also showed that updating the centers using a gradient descent based algorithm makes them move toward the majority of the data. Thereafter, the centers in RBFNNs have been commonly solved by a clustering method. First, only fully unsupervised clustering of input items (input clustering) was applied, but soon some supervision was brought also to the clustering step. In the commonly used input-output clustering approach, the input vectors are concatenated with the target output vectors and the clustering is performed on these concatenated vectors. We propose to introduce supervision into RBFNN training simply by clustering items class-by-class. We call this the class-specific approach. All these clustering approaches are introduced in detail in Section III.

Besides clustering, the RBF center locations can be determined e.g. by supervised vector quantization algorithms [10] or supervised training of decision trees [10]. In [14], the RBF center locations and widths are solved along with the number of centers using hybrid forward algorithm (HFA), which tackles the mixed integer hard problem of simultaneous network structure determination and parameter optimization on the continuous parameter space using an integrated analytic framework.

Once clustering has been performed, a RBF neuron is placed at each cluster centroid. Next, one needs to assign neuron widths and the network weights. Often acceptable classification results can be obtained even by using the same widths for all the centroids [15]. However, using a single width value may harm the performance if there is reason to assume that the RBF centroids correspond to clusters with varying sizes. This may be the case, when the class-specific clustering is applied as the distributions of different classes may be quite different. Varying width values for each RBF neuron can be obtained using a simple heuristic [16], [5], where, for each center, the distance to the nearest other cluster center is computed and multiplied by γ.

$$\sigma = \gamma \min_{j \neq i} (\|\boldsymbol{\mu}_j - \boldsymbol{\mu}_i\|). \quad (2)$$

If the class-specific clustering is used, only the clusters belonging to different classes can be considered when computing the distance. In [5], $\gamma = 5$ was applied. A few more possible ways to find a simple center-specific width value are given in [10].

In the second learning phase the weights are determined. They are typically solved either using back-propagation [16], [11] or linear least squares technique [15]. As the only parameters to be set are the weights between the RBF and output neurons, back-propagation reduces to a linear regression task.

The linear least squares technique aims at minimizing the sum of squared errors between the obtained outputs, y_{ij}, and target outputs, t_{ij}, computed over N training items and C output neurons

$$SE = \sum_{i=1}^{N} \sum_{j=1}^{C} \|y_{ij} - t_{ij}\|^2 = \|\mathbf{Y} - \mathbf{T}\|^2 = \|\mathbf{G}\mathbf{W} - \mathbf{T}\|^2, \quad (3)$$

where $\mathbf{Y} = [y_{ij}]$ and $\mathbf{T} = [t_{ij}]$ are $N \times C$ matrices, $\mathbf{G} = [g_k(\mathbf{x}_i)]$ is a $N \times K$ matrix and g_k is as defined in Eq. (1), $\mathbf{W} = [w_{kj}]$ is a $K \times C$ matrix and w_{kj} is the weight between the k^{th} RBF neuron and j^{th} output neuron. The weights are solved as

$$\mathbf{W} = [\mathbf{G}^T\mathbf{G}]^{-1}\mathbf{G}^T\mathbf{T}. \quad (4)$$

The problem with the above equation is that, if $\mathbf{G}^T\mathbf{G}$ is singular, it does not have a unique solution and with close to singular values the solution is not stable. To overcome these problems, in [15], a regularized least squares solutions was suggested:

$$\mathbf{W} = [\mathbf{G}^T\mathbf{G} + \lambda\mathbf{G}_0]^{-1}\mathbf{G}^T\mathbf{T}, \quad (5)$$

where λ is a regularization constant and $\mathbf{G}_0 = [g_i(\boldsymbol{\mu}_j)]$ is a symmetric $C \times C$ square matrix whose elements are the outputs of RBF units when the RBF centers are given as inputs. With the regularized least squares technique, the choice of a proper value for the regularization constant λ is important, but far from simple [17].

There are also several training methods that do not fit to the traditional categorization of one-, two-, or three-phase learning. Back-propagation or other gradient descent methods used in the third-phase of the three-phase learning can be applied also with a random initialization [16], [11], [18] i.e. without any other training. A limitation with back-propagation is that it cannot evolve the network architecture, but it must be fixed a priori. Evolutionary and genetic learning algorithms [19], [20] can simultaneously search for all the network parameters including the network architecture, but these methods are generally very slow.

In the recent years, extreme learning [21] and different sequential training methods [22] have obtained very promising

results in the RBFNN training. In the sequential methods, the RBF centers are selected from the training items presented one-by-one by applying different growing and pruning strategies. Also the other network parameters are generally updated item-by-item, if the new item provides some new information to the network. In [23], it has been experimentally shown that for the sequential methods it is beneficial to apply also class-specific criteria and not only criteria based on overall approximation error.

Other advances in the area of RBFNN training include optimization of the widths of the Gaussian neurons [24], metacognitive learning [22], and second order training methods [25]. These methods either apply or can be applied with clustering. In this article, we are not trying to present a new state-of-the-art RBFNN training method, but we only consider different clustering approaches with otherwise simple training steps to clearly see the significance of the compared approaches. However, in the future, the comparison results can be exploited with such state-of-the-art techniques to further improve them. Similarly, the results can be also exploited with modifications of the basic RBFNNs such as hybrid radial basis function and data envelopment analysis neural networks [26], polynomial RBFNNs [27], or networks with different radial functions and shape parameters [28].

III. Different Clustering Approaches in RBFNN Training

A. Input Clustering

The input clustering refers to the basic unsupervised clustering, where only the training inputs are clustered without considering the available class information. This kind of clustering has been widely used in RBFNN training e.g. in [15], [29], [30].

B. Input-output Clustering

In the input-output clustering, the vectors to be clustered, $\mathbf{x}_i^{IO}$ are formed by concatenating the input vectors with the target output vectors weighted by a factor β:

$$\mathbf{x}_i^{IO} = [\mathbf{x}_i, \beta \mathbf{t}_i], \tag{6}$$

where $\mathbf{x}_i$ is the i^{th} input vector in the training set and $\mathbf{t}_i$ is the corresponding target output vector. The weight is traditionally set so that each element in the augmented vector will have a similar order of magnitude [31]. Such a weighting will naturally place more emphasis on the output part, if the input feature vector dimension is lower or the number of classes is higher (longer 1-of-C-encoded output vector). After the clustering phase, the outputs are discarded and the RBF neuron centers are set in the input space. It has been shown that for a suitable weighting factor the training error can be made arbitrarily small by choosing a sufficiently large number of hidden neurons [32], but naturally this may lead to poor generalization ability and an undesired network structure.

While the input-output clustering is assumed to lead to a better performance than the input clustering, only few comparisons have been carried out to compare the approaches. In [31], the input-output clustering approach gave a better performance as a dynamic model for a time-series system, but in this comparison also different RBF center widths were assigned for the two approaches. In [33], the input-output clustering outperformed the input clustering in function prediction on two datasets. We are not aware of any comparisons on classification tasks.

Several modifications or extensions of the basic input-output clustering have been proposed for RBFNN training [33], [34], [35]. Also several supervised clustering techniques, which perform clustering in a single process, but take the class information somehow into account, can be considered as input-output clustering. In [36], constraints based on class information were placed for the fuzzy C-means algorithm. In [37], a similar modification of the fuzzy C-means algorithm was applied, but now the objective function was modified to take into account also class information. In [35], a new network type, radial basis perceptron (RBP) neural network, was introduced and trained with a specific input-output clustering method. In [10], supervised vector quantization was performed by considering also the class information in the applied learning rule. In [38], clustering was done using a minimum spanning tree (MST). All the training items were first considered as vertices of a MST and the edges were weighted with the squared Euclidean distances of the items. All the edges connecting items from different classes were then removed and the remaining connected components thus represented clusters having items from a single class only. However, all these methods require using the specific clustering algorithm, while the basic input-output clustering approach can be easily used with any available clustering algorithm.

C. Class-specific Clustering

We propose to introduce supervision into RBFNN training simply by clustering items class-by-class. We call this approach class-specific clustering. Also clustering-inside-class or within-class clustering are used to refer to the same approach. We are motivated by several successful divide-and-conquer methods, which obtain a better final solution by dividing a complicated problem into several sub-problems. In this case, performing the clustering operation per class will make the task significantly easier and more efficient. This is due to the fact that there are fewer local optima. Clustering will be also significantly faster due to the smaller number of items. Also each cluster centroid now corresponds to a single class.

In this paper, no post-processing of the centroids is considered. Even if two centroids of two different classes happen to be co-located, they will both be used as RBF centers as such. For overlapping data distributions, the centroids may naturally be close to each other and such a case could be directly used to indicate that the area should be modeled with more RBF centers. A possible rule for handling overlapping clusters from different classes is presented in [6]. However, this is left for further work to clearly see the impact of the simplest versions of different clustering approaches.

The idea of the class-specific clustering in the RBFNN training has been previously used at least in [5], [6], [39], [40]. In [39], [40], and [5], APC-III clustering method is

used to select RBF neuron locations separately for each class. All these works concentrate on a single handwritten digit recognition dataset and the significance of the class-specific clustering compared to the traditional clustering approaches is not evaluated. In [6], the RBF centers are initially assigned to the centroids of each class and all the items are assigned to these clusters. Then an iterative method is applied, where the overlap between clusters from different classes is evaluated and the clusters are further divided if they are detected to overlap. The paper concentrates on a single face recognition dataset. The applied clustering method is compared with unsupervised K-means, but only in terms of clustering error. K-means is not used to initialize RBF centers. The training items are handled class-by-class also in [41], but, instead of clustering, a density estimation algorithm is applied and the significance of the class-specific approach is not evaluated.

D. On Time and Memory Complexity

The time and memory complexity of most clustering algorithm is dependent on the number of items N, number of clusters K, and the data dimensionality d (e.g. $O(NKd)$ for a linear algorithm). For the input and class-specific clustering approaches d is directly the dimensionality of the input feature vectors, whereas for the input-output clustering approach the data is formed by concatenating the input features vectors and target output vectors i.e. $d_{IO} = d_I + C$, where IO and I refer to the input-output and input clustering approaches and C is the number of classes. The class-specific clustering approach must be repeated C times, but the number of items and clusters per clustering is smaller. If it is assumed that the items are equally distributed between classes and that the total number of clusters is the same and an equal number of clusters is assigned for each class, $N_{CS} = N_I/C$ and $K_{CS} = K_I/C$, where CS refers to the class-specific clustering approach.

The effect of the above changes naturally depends on the complexity of applied clustering algorithm as well as the data properties. For most real-life benchmark datasets $d_{IO} \approx d_I$, but for some datasets $d_{IO} > 2d_I$. In the latter situation, the input-output clustering would take more than twice longer than the input clustering approach even when a clustering algorithm with a linear complexity is applied. If the clustering algorithm is not otherwise affected by the clustering approach (e.g. number of iterations), the class-specific approach is C times faster if an algorithm with linear complexity is applied and more if a slower algorithm is used. In the next section, we will shortly consider the complexity of the applied clustering algorithms as examples.

IV. Applied Clustering Algorithms

A. K-means

K-means is the most widely-used and well-known clustering algorithm. Therefore, we will not introduce it here. A description can be found e.g. in [42]. The popularity of the algorithm can be explained by its speed and the relatively good results obtained using it. However, as a heuristic algorithm K-means will converge to the any encountered local optimum and, therefore, initialization of the algorithm may affect the results significantly. Its main drawback is that the number of clusters must to be assigned a priori.

The time complexity of the K-means algorithm is $O(iNKd)$. In practice, the algorithm often converges in tens of iterations and, therefore, the practical complexity is close to linear. However, the theoretical upper bound for the expected number of iterations is $O(\frac{N^{34}\log^4 NK^{34}d^8}{\sigma^6})$, where σ is the standard deviation of the Gaussian perturbations [43].

B. APC-III

APC-III is a simple one-pass clustering algorithm [5], [39], which solves also the number of clusters during the clustering process. It has been previously applied for RBFNN training via the class-specific clustering in [5], [39], [40] and also via the input clustering in [44], [45] , but the two approaches have not been compared.

All the items are considered one by one. If an item is within a defined cluster radius, R_0, of the centroid of any existing cluster, the item is added to that cluster by adjusting its centroid accordingly. Otherwise, a new cluster is created and the item is assigned as its first centroid. This way the data space will always be divided into clusters of fixed size even if the data distribution would indicate totally different true clusters. The algorithm is further described in Algorithm 1.

input : $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ – datasamples
output : K – number of clusters
$\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_K$ – cluster centroids
variables: R_0 – cluster radius
N_i – number of items in the i^{th} cluster
$d_i j$ – distance between $\mathbf{x}_i$ and $\boldsymbol{\mu}_j$

Compute $R_0 = \alpha \frac{1}{N} \sum_{i=1}^{N} \min_{i \neq j}(\|\mathbf{x}_i - \mathbf{x}_j\|)$
Set $K = 1, \boldsymbol{\mu}_1 = \mathbf{x}_1, N_i = 1$
for $i := 2 \rightarrow N$ **do** `// For each item`
  for $j := 1 \rightarrow K$ **do** `// For each cluster`
    Compute d_{ij}
    if $d_{ij} \leq R_0$ **then**
      `// Include` $\mathbf{x}_i$ `into` j^{th} `cluster`
      $\boldsymbol{\mu}_j \leftarrow (\boldsymbol{\mu}_j N_j + \mathbf{x}_i)/(N_j + 1)$
      $N_j \leftarrow N_j + 1$
      Exit the for-loop
    end
  end
  if $\mathbf{x}_i$ *was not included into any cluster* **then**
    `// Create a new cluster`
    $K \leftarrow K + 1$
    $\boldsymbol{\mu}_K \leftarrow \mathbf{x}_i$
    $N_K \leftarrow 1$
  end
end

Algorithm 1: APC-III algorithm

The result of APC-III is quite dependent on the presentation sequence of the items and the number of clusters produced heavily depends on parameter α. As a one pass algorithm it has the complexity of $O(NKd)$.

C. Multi-dimensional Particle Swarm Optimization and Fractional Global Best Formation

Recently, MD PSO and FGBF have been successfully used for dynamic clustering e.g. in [46]. As a clustering method based on stochastic optimization, the method can avoid local optima and is not dependent on the starting point. Clustering based on MD PSO and FGBF was further applied for the RBFNN training e.g. in [47], but only the input or input-output clustering approaches have been used. To the best of our knowledge, the combination of an evolutionary dynamic clustering method and the class-specific clustering approach has not been proposed before.

In the following subsections, we will first describe the basic particle swarm optimization (PSO) algorithm, then its extension MD PSO, and the FGBF plug-in. Finally, in Section IV-C4 we describe how to apply the methods for clustering.

1) Particle Swarm Optimization: The basic form of the PSO algorithm was introduced by J. Kennedy and R. Eberhart in [48] and later modified by Shi and Eberhart in [49]. In the algorithm, a swarm of S particles flies through an D-dimensional search space where each particle's position represents a potential solution to an optimization problem. Each particle p with current position $\mathbf{a}_p$ and current velocity $\mathbf{v}_p$ remembers its personal best solution so far, $\mathbf{b}_p$. The swarm as a whole remembers the overall best solution globally achieved so far, $\mathbf{b}_S$.

At every iteration t, a fitness function value is first computed for all particle positions. The personal best solutions, $\mathbf{b}_p$, for all particles and the global best solution, $\mathbf{b}_S$, are updated if necessary. Then the following velocity and position updates are applied on each particle p:

$$\begin{aligned}\mathbf{v}_p(t+1) =& w(t)\mathbf{v}_p(t) + c_1\mathbf{r_1}(t)\left(\mathbf{b}_p(t) - \mathbf{a}_p(t)\right) + \\ & c_2\mathbf{r_2}(t)\left(\mathbf{b}_S(t) - \mathbf{a}_p(t)\right) \\ \mathbf{a}_p(t+1) =& \mathbf{a}_p(t) + \mathbf{v}_p(t+1),\end{aligned} \tag{7}$$

where $w(t)$ is an inertia weight, c_1 and c_2 are acceleration constants, which are usually set to 2, and $\mathbf{r_1}(t)$ and $\mathbf{r_2}(t)$ are vectors of random variables uniformly distributed between 0 and 1. A larger value of $w(t)$ favors exploration while smaller inertia weight values favor exploitation. Often $w(t)$ is linearly decreased from a high value (e.g. 0.75) to a low value (e.g. 0.2) during the iterations of a PSO run.

The random initialization and the random variables $\mathbf{r_1}$ and $\mathbf{r_2}$ make PSO a stochastic search method. Thus, it can avoid some local optima. However, for the basic form of the PSO algorithm, premature convergence to a local optimum is still a common problem.

2) Multi-dimensional Particle Swarm Optimization: MD PSO [46] is an extension of the basic PSO algorithm where particles can search for optimal solutions with different dimensionalities within a given dimensionality range, $\{d_{\min}, ..., d_{\max}\}$. For example, in the clustering case this would mean that the algorithm can be used to search for the optimal number of clusters simultaneously to searching for the optimal cluster centroid locations. In order to accomplish this, each particle has two sets of components, each of which has been subjected to one of the two independent and consecutive PSO processes. The first one is the regular positional PSO, which takes place in each particle's current dimensionality, d_p, and the second one is a dimensional PSO, which allows particles to move among search spaces with different dimensionalities. For each dimensionality d, the swarm keeps track of the best global position so far achieved, $\mathbf{b}_S^d$, and each particle p keeps track of its last position $\mathbf{a}_p^d$, velocity $\mathbf{v}_p^d$, and personal best position $\mathbf{b}_p^d$ in that particular dimensionality so that when it re-visits the same dimensionality at a later time, it can perform the regular positional updates using this information. The dimensional PSO process may then again move a particle to another dimensionality, where it will remember its positional status and will be updated through the positional PSO process, and so on.

The positional PSO updates take place in each particle's current dimensionality, d_p, according to Eq. (7). The particle's new position $\mathbf{a}_p^{d_p}(t+1)$ still has the same dimensionality d_p. However, the dimensional PSO process may move the particle into another dimensionality. The search will continue from $\mathbf{a}_p^{d_p}(t+1)$ if the particle later returns to dimensionality $d_p(t)$.

Similarly to the positional PSO, the dimensional PSO process uses the personal best dimensionality of each particle, db_p, and the global best dimensionality, db_S, to attract the particles toward a better dimensional solution. The dimensional PSO updates are:

$$\begin{aligned}dv_p(t+1) =& \lfloor dv_p(t) + c_1 r_1(t)\left(db_p(t) - d_p(t)\right) + \\ & c_2 r_2(t)\left(db_S(t) - d_p(t)\right)\rfloor \\ d_p(t+1) =& d_p(t) + dv_p(t+1),\end{aligned} \tag{8}$$

where dv_p is the dimensional velocity of particle p and $\lfloor\cdot\rfloor$ denotes the floor operator. Finally, the global best dimension, db_S, and the global best solution with that dimensionality, $\mathbf{b}_S^{db_S}$, represent the optimal dimensionality and the optimal solution respectively. Further details and the pseudo-code of MD PSO can be found in [46] and [50].

3) Fractional Global Best Formation: Both the basic PSO algorithm and its multi-dimensional extension, MD PSO, may suffer from premature convergence to a local optimum particularly when a high dimensional optimization problem with a multi-modal fitness surface is encountered. The premature convergence is mainly caused by a loss of diversity i.e. the particles gather too close to $\mathbf{b}_S$ in an early phase and lose their ability to explore new potential solutions. The FGBF process presented in [46] can efficiently address the premature convergence problem. The main idea of FGBF is to create at every iteration an additional artificial solution $\mathbf{b}_A$ by combining the best individual elements of particles' solutions. This artificial solution is then compared to $\mathbf{b}_S$ and, if it turns out to have a higher fitness value, $\mathbf{b}_A$ will replace $\mathbf{b}_S$ in Eq. (7). If the solution $\mathbf{b}_A$ is not better than $\mathbf{b}_S$, the PSO process will proceed as usual. To decide which elements of the particles' solutions should be combined when creating $\mathbf{b}_A$, the elements are evaluated with a fractional fitness function. The fractional fitness function for FGBF should be specifically designed for each optimization problem similarly to designing the fitness function for MD PSO.

When FGBF is used in combination with MD PSO, a sepa-

rate artificial solution, $\mathbf{b}_A^d$, is created for every dimensionality d within the dimensionality range and in every dimensionality $\mathbf{b}_A^d$ competes with $\mathbf{b}_S^d$. Depending on the optimization task, during the formation of $\mathbf{b}_A^d$ it may be possible to combine elements from particle positions with different dimensionalities.

4) Clustering with MD PSO and FGBF: In clustering, similar to other PSO applications, each particle's position should represent a potential solution to the problem. Most often this is realized in clustering by encoding the position of particle p as $\mathbf{a}_p = \{\boldsymbol{\mu}_{p,1}, ..., \boldsymbol{\mu}_{p,j}, ..., \boldsymbol{\mu}_{p,K}\}$, where $\boldsymbol{\mu}_{a,j}$ represents the j^{th} (potential) cluster centroid in an D-dimensional data space and K is the number of clusters [51]. When MD PSO is used for clustering, also K can be optimized via the dimensional PSO process. Any given clustering validity index function (CVI) can be used as the fitness function.

MD PSO and FGBF were first applied to clustering in [46]. The particle encoding used is a straightforward multi-dimensional extension of the centroid-based encoding used in [51], i.e. $\mathbf{a}_p^d = \{\boldsymbol{\mu}_{p,1}, ..., \boldsymbol{\mu}_{p,j}, ..., \boldsymbol{\mu}_{p,d}\}$. Now each particle has a position in every dimensionality $d \in \{d_{\min}, ..., d_{\max}\}$ and, in the d^{th} dimensionality, particle positions represent d potential cluster centroids. The dimensional PSO process leads particles towards solutions with a better cluster number while the interleaved positional PSO process helps to find better centroids for a certain number of clusters.

The objective of the FGBF process is to combine the best individual elements (cluster centroids in this case) to create artificial solutions $\mathbf{b}_A^d$ to compete with $\mathbf{b}_S^d$ solutions. However, finding the best cluster centroid combinations among all the possibilities is not simple, since in clustering the quality of a certain centroid always depends on the other centroids. It is not feasible to consider all the possible centroid combinations, but if the quality of a certain centroid is evaluated purely locally with respect to the data items without considering other centroids, then, it is likely that only undesired solutions would be obtained. It should be remembered, however, that the FGBF operation is only used as a means to avoid premature convergence. If FGBF cannot form competent solutions, it will not have any effect on MD PSO as the artificial solutions will not be used.

To ensure that at least one data item is assigned to each cluster, the FGBF operation uses only the centroids which are closest to at least one data item. To avoid selecting centroids representing the same natural cluster, centroids are first grouped into centroid groups using a MST [52]. A certain number of centroid groups, say $d \in \{d_{\min}, ..., d_{\max}\}$, can be obtained from the MST simply by breaking the $d-1$ longest MST branches. The artificial solution $\mathbf{b}_A^d$ is then formed by choosing only one centroid form each group. Note that the solution elements (centroids) can now be combined regardless of the dimensionality (number of clusters) of the original particle positions.

We use Xu CVI [53] as the fitness function for MD PSO. It attempts to minimize the within-cluster sum-of-squares, ssw, defined as

$$ssw = \sum_{i=1}^{K} \sum_{\mathbf{x} \in c_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2, \tag{9}$$

where $\boldsymbol{\mu}_i$ is the centroid of i^{th} cluster c_i, $\mathbf{x}$ is a data item, and K is the number of clusters. The Xu CVI is defined as

$$Q(\mathbf{a}_p^d) = D \log\left(\sqrt{\frac{ssw}{DN^2}}\right) + \log K, \tag{10}$$

where D is the data dimension and N is the total number of data items. The fractional fitness function for the FGBF is defined as

$$Q(\mathbf{a}_{p,j}^d) = Q(\boldsymbol{\mu}_{p,j}) = \sum_{i=1}^{M} \frac{\|\mathbf{x}_{[i]} - \boldsymbol{\mu}_{p,j}\|}{M}, \tag{11}$$

where $\boldsymbol{\mu}_{p,j}$ is the centroid defined by the j^{th} element of particle p's position in dimensionality d and $\mathbf{x}_{[i]}$ represents the i^{th} closest data item to $\boldsymbol{\mu}_{p,j}$. For computational simplicity, only data items assigned to cluster c_j defined by $\boldsymbol{\mu}_{p,j}$ in solution $\mathbf{a}_p^d$ are considered. $M = \min(10, N_j)$, where N_j is the number of data items assigned to cluster c_j in solution $\mathbf{a}_p^d$.

The computational complexity of the MD PSO algorithm is mainly caused by the fitness function evaluations, as they must be performed for every particle at every iteration. In this work, a fixed iteration number is used and thus the complexity is of form $O(SO(FF))$, where S is the swarm size and $O(FF)$ is the complexity of the fitness function. The complexity of the Xu CVI is of form $O(NKd_{data})$, where d_{data} is the dimensionality of the input items and K is at highest $d_{\max}$ assigned for MD PSO. The FGBF operation is performed once per iteration. It requires, first of all, evaluating, which fractional solutions are closest to some data item. This evaluation is of complexity $O(SKNd_{data})$. Then artificial solutions are created, which requires forming a MST. Computational complexity of the algorithm applied for MST formation is $O(n \log n)$, where n refers to the number of considered fractional solutions. In practice, it takes less time than the rest of the FGBF operation even for the largest considered datasets. After creating the artificial solutions, they must be evaluated using the fitness measure. These evaluations are of complexity $O((d_{\max} - d_{\min})NKd_{data})$. When the class-specific clustering approach is applied, $d_{\max}$ can be divided by C and the average number of items considered per clustering is also divided by C. Thus, computation time will be significantly reduced in different parts of MD PSO. Repeating the clustering operation for each class is a minor cost in comparison to all the savings and, also in practice, significant reductions in computation times are observed.

V. Experimental Setup and Results

A. Experimental Setup and Parameters

We compare the class-specific, input, and input-output clustering approaches for RBFNN training using K-means, APC-III, and MD PSO algorithms. In each case, the rest of the training is carried out in a similar way. After the RBF center locations have been determined via clustering, the center widths are computed according to Eq. (2). The parameter γ is set to 5 and the class information is not considered even for the class-specific approach. The weights are determined by solving Eq. (4) using functions provided by ALGLIB numerical analysis and data processing library [54]. The linear

least squares problem is solved using a numerical iterative algorithm LSQR, which has been observed to usually provide acceptable solutions also for singular problems [55].

We evaluate the results mainly in terms of the average classification accuracy. As the overall classification accuracy is not properly evaluating the performance, when the class distribution is highly imbalanced, we also compute the geometric mean of recall values for each class [56]:

$$G-mean = \left(\prod_{i=1}^{C} R_i\right)^{1/C} = \left(\prod_{i=1}^{C} \frac{n_{ii}}{\sum_{j=1}^{K} n_{ji}}\right)^{1/C}, \quad (12)$$

where n_{ii} and n_{ji} refer to the elements of the corresponding confusion matrix. It should be remembered, however, that the network weights have been assigned by minimizing the sum of squared errors between the obtained outputs and target outputs computed over all N training items and C output neurons. This also aims at minimizing the overall error. Therefore, the computed G-mean values cannot directly evaluate the ability of different clustering approaches to handle imbalanced class distribution, but also the rest of training process should be modified accordingly if this was the main objective.

For K-means, we used three different total numbers of cluster, $3C$, $\max(5C, \sqrt{N})$, and $N/3$. For the input and input-output clustering approaches K was directly set to these numbers, while the numbers were divided by the number of classes, C, for the class-specific approach. For APC-III, the only parameter α was set to 1.5. This value was used also in [5], when comparing with other methods.

For MD PSO the total number of clusters was allowed to vary from C to $\max(5C, \sqrt{N})$. For the class-specific clustering the numbers were divided by C. The parameters for MD PSO were set as follows: The swarm size, S, was set to 100 and the number of iterations to 200. The dimensionality ranges, $\{d_{\min}, ..., d_{\max}\}$, were set according to the cluster number ranges specified above and the dimensional velocity range, $\{dv_{\min}, ..., dv_{\max}\}$, to $\{-2, ..., 2\}$ for all the experiments. The positional search space range $\{\mathbf{a}_{\min}, ..., \mathbf{a}_{\max}\}$ was set according to the data values as $\{0, ..., 1\}$. The positional velocity range, $\{\mathbf{v}_{\min}, ..., \mathbf{v}_{\max}\}$, was set to $\{-0.1, ..., 0.1\}$. The inertia weight $w(t)$ was linearly decreased from 0.75 to 0.2 during the MD PSO iterations. The acceleration constants c_1 and c_2 in Eq. (7) and (8) were set to 2. In [15], the MD PSO parameters were analyzed and the algorithm was tested not be sensitive to parameter changes. Naturally, the swarm size and iteration number could be adjusted according to the difficulty of the problem. However, in this paper we did not aim at the maximal performance, but the objective was to compare different clustering approaches under equal conditions.

The clustering approaches were compared using random 5-fold cross validation on each dataset. As the applied clustering methods were stochastic, we created two different divisions into folds and repeated the training process 10 times on each training subset (i.e. each clustering approach was evaluated 100 times on each dataset). An exception was made with K-means and $3C$ clusters. Due to the higher standard deviations observed, we repeated the experiment additional 40 times on each training subset (i.e. a total of 500 evaluations). The weight β used to scale outputs for the input-output clustering was set to 1. The experiments were carried out using the Merope grid of Tampere Center for Scientific Computing [57].

B. Datasets

Evaluations were performed on 25 benchmark classification datasets from UCI Machine Learning Repository [58]. To reduce the probability of outliers affecting the results significantly, we removed all the classes containing less than 20 items. Thus all the datasets contain at least four items from each class in each fold. The characteristics of the datasets are shown in Table I. In the class column, we show the number of used classes over the total number of classes. All the data values were normalized between 0 and 1. The outputs are given using 1-of-C encoding i.e. the output [0 1 0] means that the item belongs to the second class in a three-class dataset.

Table I
CHARACTERISTICS OF THE DATASETS OBTAINED FROM UCI REPOSITORY

Dataset	No. of objects	Dimensionality	No. of classes
Banknote auth.	1372	4	2/2
Breast Wisconsin	569	30	2/2
Dermatology	366	33	6/6
Ecoli	336	7	5/8
Haberman	306	3	2/2
Heart Cleveland	297	13	5/5
Ionosphere	351	34	2/2
Iris	150	4	3/3
Letter recognition	20000	16	26/26
Magic G. Telescope	19020	11	2/2
Movement libras	360	90	15/15
Musk	476	166	2/2
Parkinsons	195	22	2/2
Pima Ind. Diabetes	768	8	2/2
Seeds	210	7	3/3
Segmentation	2310	19	7/7
Sonar all	208	60	2/2
Spectf	267	44	2/2
Transfusion	748	4	2/2
Vehicle	846	18	4/4
Vertebral column	310	6	3/3
Vowel context	990	10	9/11
Wine	178	13	3/3
Winequality red	1599	11	4/6
Yeast	1484	8	9/10

C. Results of the Comparative Evaluations

The comparison results using MD PSO are given in Table II, using APC-III in Table III, and using K-means with different values of K in Tables IV-VI. *CS*, *I*, and *IO* refer to the class-specific, input, and input-output clustering approaches. The results are averages over all the folds and runs and the standard deviations are reported next to accuracies and geometric mean values. In Table VII, we give the total cluster numbers produced by different clustering approaches when MD PSO and APC-III methods were used. In Table VIII, iteration numbers required for completion of the K-means algorithm, when the total number of clusters is $3C$ or $\max(5C, \sqrt{N})$, are given. For the class-specific approach we show the total numbers of iterations required for all the classes together, but in brackets we also show the average iteration number per class for easier comparison.

The results clearly indicate that the class-specific clustering approach is superior to the other approaches, when MD PSO,

Table II
CLASSIFICATION ACCURACIES AND GEOMETRIC MEAN VALUES OF RBFNNS TRAINED USING MD PSO FOR CLUSTERING

	Accuracy			G-mean		
Dataset	CS	IO	I	CS	IO	I
Banknote auth.	0.990 ± 0.006	$\mathbf{0.996 \pm 0.002}$	0.993 ± 0.004	0.991 ± 0.005	$\mathbf{0.996 \pm 0.002}$	0.994 ± 0.004
Breast Wisconsin	$\mathbf{0.959 \pm 0.007}$	0.956 ± 0.010	0.957 ± 0.008	$\mathbf{0.945 \pm 0.010}$	0.944 ± 0.013	0.944 ± 0.011
Dermatology	$\mathbf{0.976 \pm 0.006}$	0.966 ± 0.017	0.969 ± 0.012	$\mathbf{0.972 \pm 0.007}$	0.959 ± 0.025	0.964 ± 0.017
Ecoli	0.872 ± 0.005	0.867 ± 0.020	$\mathbf{0.882 \pm 0.008}$	0.797 ± 0.009	0.732 ± 0.102	$\mathbf{0.807 \pm 0.011}$
Haberman	$\mathbf{0.747 \pm 0.002}$	0.744 ± 0.005	0.741 ± 0.007	$\mathbf{0.371 \pm 0.016}$	0.265 ± 0.084	0.074 ± 0.085
Heart Cleveland	0.599 ± 0.013	0.619 ± 0.015	$\mathbf{0.622 \pm 0.016}$	0.058 ± 0.062	$\mathbf{0.131 \pm 0.055}$	0.105 ± 0.088
Ionosphere	$\mathbf{0.909 \pm 0.011}$	0.896 ± 0.013	0.884 ± 0.049	$\mathbf{0.875 \pm 0.017}$	0.849 ± 0.020	0.832 ± 0.071
Iris	$\mathbf{0.970 \pm 0.006}$	0.966 ± 0.013	0.957 ± 0.019	$\mathbf{0.969 \pm 0.006}$	0.964 ± 0.014	0.955 ± 0.020
Letter recognition	$\mathbf{0.741 \pm 0.005}$	0.361 ± 0.102	0.578 ± 0.014	$\mathbf{0.725 \pm 0.005}$	0.010 ± 0.032	0.474 ± 0.077
Magic G. Telescope	$\mathbf{0.819 \pm 0.024}$	0.710 ± 0.052	0.690 ± 0.044	$\mathbf{0.760 \pm 0.031}$	0.471 ± 0.243	0.386 ± 0.232
Movement libras	$\mathbf{0.788 \pm 0.015}$	0.758 ± 0.022	0.758 ± 0.021	$\mathbf{0.487 \pm 0.139}$	0.427 ± 0.144	0.408 ± 0.111
Musk	$\mathbf{0.782 \pm 0.025}$	0.749 ± 0.027	0.749 ± 0.030	$\mathbf{0.774 \pm 0.027}$	0.738 ± 0.029	0.738 ± 0.032
Parkinsons	$\mathbf{0.857 \pm 0.012}$	0.856 ± 0.012	0.848 ± 0.020	$\mathbf{0.710 \pm 0.020}$	0.707 ± 0.008	0.696 ± 0.032
Pima Ind. Diabetes	$\mathbf{0.756 \pm 0.009}$	0.691 ± 0.021	0.701 ± 0.020	$\mathbf{0.684 \pm 0.011}$	0.523 ± 0.068	0.574 ± 0.041
Seeds	0.948 ± 0.007	0.940 ± 0.012	$\mathbf{0.949 \pm 0.007}$	0.946 ± 0.007	0.938 ± 0.013	$\mathbf{0.947 \pm 0.007}$
Segmentation	$\mathbf{0.902 \pm 0.005}$	0.877 ± 0.035	0.898 ± 0.005	$\mathbf{0.893 \pm 0.006}$	0.859 ± 0.048	0.888 ± 0.006
Sonar all	$\mathbf{0.751 \pm 0.031}$	0.734 ± 0.038	0.738 ± 0.040	$\mathbf{0.747 \pm 0.032}$	0.729 ± 0.039	0.732 ± 0.041
Spectf	$\mathbf{0.819 \pm 0.016}$	0.810 ± 0.020	0.804 ± 0.018	$\mathbf{0.583 \pm 0.044}$	0.515 ± 0.058	0.438 ± 0.062
Transfusion	0.782 ± 0.007	0.785 ± 0.005	$\mathbf{0.788 \pm 0.007}$	0.432 ± 0.033	0.465 ± 0.020	$\mathbf{0.472 \pm 0.022}$
Vehicle	0.737 ± 0.015	$\mathbf{0.743 \pm 0.010}$	0.742 ± 0.016	0.702 ± 0.019	$\mathbf{0.712 \pm 0.012}$	$\mathbf{0.712 \pm 0.018}$
Vertebral column	$\mathbf{0.818 \pm 0.014}$	0.810 ± 0.020	0.810 ± 0.023	$\mathbf{0.769 \pm 0.020}$	0.735 ± 0.044	0.742 ± 0.052
Vowel context	$\mathbf{0.722 \pm 0.017}$	0.671 ± 0.027	0.668 ± 0.041	$\mathbf{0.642 \pm 0.026}$	0.555 ± 0.049	0.541 ± 0.075
Wine	0.987 ± 0.004	0.899 ± 0.046	$\mathbf{0.989 \pm 0.004}$	0.988 ± 0.003	0.901 ± 0.046	$\mathbf{0.989 \pm 0.003}$
Winequality red	$\mathbf{0.592 \pm 0.007}$	0.559 ± 0.002	0.548 ± 0.033	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Yeast	0.591 ± 0.009	0.569 ± 0.018	$\mathbf{0.592 \pm 0.011}$	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

Table III
CLASSIFICATION ACCURACIES AND GEOMETRIC MEAN VALUES OF RBFNNS TRAINED USING APC-III FOR CLUSTERING

	Accuracy			G-mean		
Dataset	CS	IO	I	CS	IO	I
Banknote auth.	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
Breast Wisconsin	$\mathbf{0.974 \pm 0.006}$	0.971 ± 0.005	0.971 ± 0.005	$\mathbf{0.968 \pm 0.007}$	0.964 ± 0.006	0.964 ± 0.006
Dermatology	$\mathbf{0.981 \pm 0.003}$	0.978 ± 0.006	0.979 ± 0.006	$\mathbf{0.978 \pm 0.004}$	0.974 ± 0.007	0.975 ± 0.006
Ecoli	0.867 ± 0.013	$\mathbf{0.871 \pm 0.009}$	0.869 ± 0.009	0.797 ± 0.017	$\mathbf{0.802 \pm 0.013}$	0.798 ± 0.013
Haberman	0.716 ± 0.016	$\mathbf{0.727 \pm 0.013}$	0.721 ± 0.014	0.473 ± 0.039	$\mathbf{0.488 \pm 0.033}$	0.482 ± 0.033
Heart Cleveland	0.571 ± 0.023	$\mathbf{0.585 \pm 0.026}$	0.572 ± 0.020	$\mathbf{0.191 \pm 0.083}$	0.181 ± 0.133	0.165 ± 0.072
Ionosphere	$\mathbf{0.921 \pm 0.014}$	0.917 ± 0.013	0.917 ± 0.013	$\mathbf{0.892 \pm 0.021}$	0.885 ± 0.019	0.886 ± 0.020
Iris	0.953 ± 0.008	$\mathbf{0.959 \pm 0.008}$	0.958 ± 0.009	0.951 ± 0.008	$\mathbf{0.957 \pm 0.009}$	0.956 ± 0.009
Magic G. Telescope	$\mathbf{0.869 \pm 0.001}$	0.867 ± 0.001	0.867 ± 0.001	$\mathbf{0.830 \pm 0.002}$	0.827 ± 0.001	0.828 ± 0.002
Movement libras	$\mathbf{0.824 \pm 0.015}$	0.804 ± 0.018	0.816 ± 0.018	$\mathbf{0.706 \pm 0.047}$	0.678 ± 0.077	0.705 ± 0.113
Musk	$\mathbf{0.885 \pm 0.016}$	0.882 ± 0.018	0.875 ± 0.021	$\mathbf{0.883 \pm 0.017}$	0.881 ± 0.017	0.873 ± 0.021
Parkinsons	0.878 ± 0.025	$\mathbf{0.883 \pm 0.025}$	0.879 ± 0.024	$\mathbf{0.797 \pm 0.038}$	0.792 ± 0.040	0.784 ± 0.031
Pima Ind. Diabetes	0.758 ± 0.012	$\mathbf{0.759 \pm 0.011}$	$\mathbf{0.759 \pm 0.011}$	0.698 ± 0.016	$\mathbf{0.703 \pm 0.014}$	0.701 ± 0.015
Seeds	0.957 ± 0.010	0.955 ± 0.009	$\mathbf{0.961 \pm 0.011}$	0.955 ± 0.011	0.953 ± 0.010	$\mathbf{0.959 \pm 0.011}$
Segmentation	$\mathbf{0.959 \pm 0.004}$	0.956 ± 0.003	0.957 ± 0.003	$\mathbf{0.958 \pm 0.004}$	0.954 ± 0.003	0.955 ± 0.003
Sonar all	$\mathbf{0.793 \pm 0.029}$	0.774 ± 0.036	0.768 ± 0.033	$\mathbf{0.787 \pm 0.029}$	0.768 ± 0.037	0.762 ± 0.034
Spectf	$\mathbf{0.829 \pm 0.017}$	0.822 ± 0.020	0.822 ± 0.019	$\mathbf{0.657 \pm 0.039}$	0.625 ± 0.039	0.625 ± 0.050
Transfusion	0.778 ± 0.009	$\mathbf{0.787 \pm 0.009}$	0.783 ± 0.009	0.548 ± 0.024	$\mathbf{0.564 \pm 0.023}$	0.559 ± 0.025
Vehicle	$\mathbf{0.829 \pm 0.012}$	0.826 ± 0.011	0.825 ± 0.011	$\mathbf{0.816 \pm 0.014}$	0.813 ± 0.013	0.812 ± 0.013
Vertebral column	0.831 ± 0.015	0.838 ± 0.012	$\mathbf{0.844 \pm 0.014}$	0.773 ± 0.017	0.777 ± 0.015	$\mathbf{0.786 \pm 0.014}$
Vowel context	$\mathbf{0.919 \pm 0.011}$	0.909 ± 0.011	0.909 ± 0.011	$\mathbf{0.900 \pm 0.013}$	0.887 ± 0.013	0.886 ± 0.014
Wine	0.988 ± 0.004	$\mathbf{0.990 \pm 0.006}$	0.989 ± 0.006	0.988 ± 0.004	$\mathbf{0.990 \pm 0.006}$	0.989 ± 0.006
Winequality red	0.604 ± 0.011	$\mathbf{0.613 \pm 0.008}$	0.611 ± 0.009	$\mathbf{0.040 \pm 0.038}$	0.029 ± 0.016	0.022 ± 0.019
Yeast	0.587 ± 0.010	$\mathbf{0.591 \pm 0.007}$	0.590 ± 0.007	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

K-means with $3C$ clusters, or K-means with $\max(5C, \sqrt{N})$ clusters are used. When APC-III or K-means with $N/3$ clusters are used, no such superiority is observed. Table VII reveals that APC-III leads to cluster numbers which are even higher than $N/3$. Thus, the first conclusion is that the class-specific clustering approach performs especially well, when a smaller amount of RBF centers is generated, but for higher cluster numbers the different clustering approaches give results closer to each other. We assume that with such high RBF center numbers their exact location is no longer that significant and, therefore, the results obtained by different clustering algorithms and approaches are closer to each other.

The differences when using APC-III and K-means with $N/3$ centroids are so small that we abstain from making further analysis on these cases, but we concentrate on the other three cases. The class-specific approach has at least 0.01 better accuracy than the other approaches on Letter recognition, Movement libras, Musk, and Vowel context datasets. All these datasets are among the most complex ones in terms of item number, dimensionality, or class number. This is an expected result since on such datasets the task of clustering the whole dataset at once is already a complicated task and, therefore, the class-specific clustering approach based on the divide-and-conquer principle has potential to yield a better performance.

The differences between the input and input-output clustering approaches are generally small. The input-output clustering

Table IV
CLASSIFICATION ACCURACIES AND GEOMETRIC MEAN VALUES OF RBFNNS TRAINED USING K-MEANS WITH $3C$ CLUSTERS IN TOTAL

	Accuracy			G-mean		
Dataset	CS	IO	I	CS	IO	I
Banknote auth.	0.960 ± 0.016	0.899 ± 0.063	$\mathbf{0.962 \pm 0.021}$	0.959 ± 0.017	0.895 ± 0.067	$\mathbf{0.964 \pm 0.021}$
Breast Wisconsin	$\mathbf{0.945 \pm 0.015}$	$\mathbf{0.945 \pm 0.016}$	0.936 ± 0.019	0.927 ± 0.020	$\mathbf{0.930 \pm 0.020}$	0.917 ± 0.027
Dermatology	$\mathbf{0.972 \pm 0.008}$	0.969 ± 0.013	0.971 ± 0.010	$\mathbf{0.968 \pm 0.009}$	0.961 ± 0.019	0.966 ± 0.012
Ecoli	$\mathbf{0.879 \pm 0.015}$	0.877 ± 0.017	$\mathbf{0.879 \pm 0.016}$	$\mathbf{0.801 \pm 0.030}$	0.793 ± 0.036	$\mathbf{0.801 \pm 0.031}$
Haberman	$\mathbf{0.750 \pm 0.015}$	0.745 ± 0.014	0.748 ± 0.012	$\mathbf{0.394 \pm 0.080}$	0.317 ± 0.110	0.348 ± 0.076
Heart Cleveland	0.611 ± 0.021	$\mathbf{0.622 \pm 0.020}$	0.617 ± 0.022	0.081 ± 0.099	$\mathbf{0.092 \pm 0.092}$	0.072 ± 0.103
Ionosphere	$\mathbf{0.893 \pm 0.022}$	0.846 ± 0.065	0.857 ± 0.049	$\mathbf{0.847 \pm 0.031}$	0.773 ± 0.102	0.795 ± 0.071
Iris	$\mathbf{0.969 \pm 0.016}$	0.960 ± 0.027	0.964 ± 0.024	$\mathbf{0.967 \pm 0.017}$	0.957 ± 0.030	0.961 ± 0.027
Letter recognition	$\mathbf{0.727 \pm 0.004}$	0.677 ± 0.008	0.690 ± 0.005	$\mathbf{0.710 \pm 0.005}$	0.645 ± 0.012	0.665 ± 0.006
Magic G. Telescope	$\mathbf{0.787 \pm 0.003}$	0.781 ± 0.019	0.757 ± 0.001	$\mathbf{0.720 \pm 0.010}$	0.713 ± 0.034	0.666 ± 0.002
Movement libras	$\mathbf{0.758 \pm 0.022}$	0.724 ± 0.026	0.729 ± 0.027	$\mathbf{0.412 \pm 0.238}$	0.278 ± 0.240	0.298 ± 0.235
Musk	$\mathbf{0.710 \pm 0.033}$	0.656 ± 0.053	0.622 ± 0.049	$\mathbf{0.700 \pm 0.036}$	0.637 ± 0.059	0.596 ± 0.057
Parkinsons	$\mathbf{0.862 \pm 0.013}$	0.835 ± 0.021	0.826 ± 0.018	$\mathbf{0.713 \pm 0.021}$	0.671 ± 0.035	0.649 ± 0.027
Pima Ind. Diabetes	$\mathbf{0.764 \pm 0.011}$	0.741 ± 0.026	0.732 ± 0.016	$\mathbf{0.697 \pm 0.015}$	0.652 ± 0.046	0.638 ± 0.027
Seeds	$\mathbf{0.937 \pm 0.019}$	$\mathbf{0.937 \pm 0.024}$	0.932 ± 0.020	$\mathbf{0.936 \pm 0.020}$	0.935 ± 0.025	0.931 ± 0.020
Segmentation	$\mathbf{0.898 \pm 0.006}$	0.891 ± 0.010	0.897 ± 0.008	$\mathbf{0.888 \pm 0.008}$	0.878 ± 0.013	0.887 ± 0.010
Sonar all	$\mathbf{0.752 \pm 0.036}$	0.747 ± 0.041	0.723 ± 0.056	$\mathbf{0.747 \pm 0.038}$	0.741 ± 0.044	0.715 ± 0.059
Spectf	$\mathbf{0.814 \pm 0.024}$	0.805 ± 0.023	0.799 ± 0.020	$\mathbf{0.525 \pm 0.089}$	0.407 ± 0.137	0.409 ± 0.118
Transfusion	$\mathbf{0.774 \pm 0.008}$	0.772 ± 0.005	0.773 ± 0.003	$\mathbf{0.308 \pm 0.073}$	0.276 ± 0.042	0.299 ± 0.026
Vehicle	$\mathbf{0.664 \pm 0.033}$	$\mathbf{0.664 \pm 0.030}$	0.648 ± 0.026	$\mathbf{0.618 \pm 0.039}$	0.608 ± 0.042	0.576 ± 0.038
Vertebral column	0.804 ± 0.025	$\mathbf{0.805 \pm 0.029}$	0.790 ± 0.024	$\mathbf{0.747 \pm 0.034}$	0.728 ± 0.048	0.728 ± 0.035
Vowel context	$\mathbf{0.657 \pm 0.024}$	0.612 ± 0.025	0.626 ± 0.023	$\mathbf{0.539 \pm 0.048}$	0.441 ± 0.089	0.475 ± 0.063
Wine	$\mathbf{0.985 \pm 0.011}$	0.973 ± 0.023	0.983 ± 0.015	$\mathbf{0.987 \pm 0.010}$	0.975 ± 0.022	0.985 ± 0.014
Winequality red	$\mathbf{0.593 \pm 0.012}$	0.587 ± 0.009	0.588 ± 0.008	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Yeast	0.585 ± 0.011	0.588 ± 0.009	$\mathbf{0.590 \pm 0.008}$	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

Table V
CLASSIFICATION ACCURACIES AND GEOMETRIC MEAN VALUES OF RBFNNS TRAINED USING K-MEANS WITH $\max(5C, \sqrt{N})$ CLUSTERS IN TOTAL

	Accuracy			G-mean		
Dataset	CS	IO	I	CS	IO	I
Banknote auth.	$\mathbf{0.997 \pm 0.003}$	$\mathbf{0.997 \pm 0.004}$	$\mathbf{0.997 \pm 0.003}$	$\mathbf{0.997 \pm 0.003}$	$\mathbf{0.997 \pm 0.003}$	$\mathbf{0.997 \pm 0.003}$
Breast Wisconsin	$\mathbf{0.961 \pm 0.009}$	0.959 ± 0.008	0.957 ± 0.009	$\mathbf{0.949 \pm 0.011}$	0.945 ± 0.011	0.943 ± 0.013
Dermatology	$\mathbf{0.975 \pm 0.007}$	0.974 ± 0.008	$\mathbf{0.975 \pm 0.008}$	$\mathbf{0.971 \pm 0.008}$	0.970 ± 0.009	$\mathbf{0.971 \pm 0.009}$
Ecoli	0.879 ± 0.010	0.881 ± 0.010	$\mathbf{0.882 \pm 0.011}$	0.805 ± 0.015	0.807 ± 0.019	$\mathbf{0.810 \pm 0.018}$
Haberman	$\mathbf{0.753 \pm 0.015}$	0.751 ± 0.014	$\mathbf{0.753 \pm 0.016}$	$\mathbf{0.485 \pm 0.039}$	0.476 ± 0.040	$\mathbf{0.485 \pm 0.044}$
Heart Cleveland	0.604 ± 0.022	$\mathbf{0.622 \pm 0.017}$	0.620 ± 0.020	0.104 ± 0.099	$\mathbf{0.129 \pm 0.070}$	0.111 ± 0.085
Ionosphere	0.919 ± 0.015	$\mathbf{0.920 \pm 0.013}$	0.916 ± 0.013	0.888 ± 0.022	$\mathbf{0.890 \pm 0.020}$	0.883 ± 0.021
Iris	$\mathbf{0.967 \pm 0.017}$	0.961 ± 0.021	$\mathbf{0.967 \pm 0.021}$	$\mathbf{0.966 \pm 0.017}$	0.959 ± 0.022	0.965 ± 0.022
Letter recognition	$\mathbf{0.757 \pm 0.003}$	0.723 ± 0.005	0.736 ± 0.004	$\mathbf{0.742 \pm 0.004}$	0.703 ± 0.006	0.718 ± 0.005
Magic G. Telescope	$\mathbf{0.853 \pm 0.002}$	0.850 ± 0.002	0.850 ± 0.002	$\mathbf{0.811 \pm 0.003}$	0.805 ± 0.003	0.806 ± 0.003
Movement libras	$\mathbf{0.781 \pm 0.021}$	0.755 ± 0.024	0.771 ± 0.023	$\mathbf{0.508 \pm 0.135}$	0.451 ± 0.191	0.497 ± 0.198
Musk	$\mathbf{0.800 \pm 0.023}$	0.790 ± 0.027	0.769 ± 0.029	$\mathbf{0.793 \pm 0.024}$	0.782 ± 0.027	0.758 ± 0.030
Parkinsons	$\mathbf{0.871 \pm 0.017}$	0.859 ± 0.016	0.864 ± 0.018	$\mathbf{0.734 \pm 0.026}$	0.716 ± 0.025	0.716 ± 0.030
Pima Ind. Diabetes	$\mathbf{0.770 \pm 0.011}$	0.768 ± 0.009	$\mathbf{0.770 \pm 0.011}$	$\mathbf{0.710 \pm 0.016}$	0.708 ± 0.013	0.708 ± 0.014
Seeds	0.945 ± 0.014	$\mathbf{0.948 \pm 0.013}$	0.945 ± 0.012	0.943 ± 0.015	$\mathbf{0.946 \pm 0.014}$	0.943 ± 0.013
Segmentation	$\mathbf{0.910 \pm 0.006}$	0.905 ± 0.005	0.905 ± 0.005	$\mathbf{0.902 \pm 0.007}$	0.896 ± 0.006	0.896 ± 0.006
Sonar all	$\mathbf{0.757 \pm 0.037}$	0.746 ± 0.033	0.745 ± 0.038	$\mathbf{0.753 \pm 0.038}$	0.742 ± 0.033	0.739 ± 0.039
Spectf	$\mathbf{0.824 \pm 0.023}$	0.823 ± 0.022	0.822 ± 0.025	$\mathbf{0.597 \pm 0.066}$	0.583 ± 0.064	0.569 ± 0.067
Transfusion	0.783 ± 0.010	0.784 ± 0.009	$\mathbf{0.786 \pm 0.009}$	$\mathbf{0.480 \pm 0.040}$	0.464 ± 0.033	0.477 ± 0.037
Vehicle	$\mathbf{0.750 \pm 0.017}$	0.749 ± 0.016	0.736 ± 0.016	0.719 ± 0.021	$\mathbf{0.721 \pm 0.021}$	0.703 ± 0.021
Vertebral column	$\mathbf{0.822 \pm 0.018}$	0.814 ± 0.014	0.811 ± 0.014	$\mathbf{0.765 \pm 0.028}$	0.751 ± 0.022	0.753 ± 0.020
Vowel context	$\mathbf{0.726 \pm 0.021}$	0.691 ± 0.021	0.702 ± 0.020	$\mathbf{0.649 \pm 0.031}$	0.588 ± 0.033	0.601 ± 0.033
Wine	$\mathbf{0.989 \pm 0.009}$	0.983 ± 0.016	0.987 ± 0.009	$\mathbf{0.990 \pm 0.008}$	0.984 ± 0.014	0.988 ± 0.008
Winequality red	$\mathbf{0.611 \pm 0.011}$	0.601 ± 0.010	0.602 ± 0.010	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Yeast	0.591 ± 0.009	0.589 ± 0.009	$\mathbf{0.593 \pm 0.009}$	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

outperforms the input clustering by 0.01 or more at least once on Ionosphere (34/2), Magic G. Telescope (11/2), Musk (166/2), Sonar all (60/2), Vehicle (18/4), Vertebral column (6/3), and Winequality red (11/4), while the input clustering similarly outperforms the input-output clustering on Banknote auth. (4/2), Ecoli (7/5), Ionosphere (34/2), Letter recognition (16/26), Movement (90/15), Pima Ind. Diabetes (8/2), Segmentation (19/7), Vowel context (10/9), Wine (13/3), and Yeast (8/9), where the dimensionality and class number are given for each dataset. Ionosphere is the only dataset appearing in both lists. Otherwise, the input-output clustering yields a better performance on datasets, where the number of classes is low, while the opposite is true for the input clustering. The input clustering outperforms the input-output clustering most clearly on Letter recognition dataset, which has 26 classes while the data dimensionality is only 16. Thus, the augmented feature vectors clustered in the input-output approach have more output than input elements. Clearly, this gives too much emphasis on the output part. It is also possible that there are more local optima when considering the augmented vectors and, therefore, on relatively complicated datasets the input clustering can find better solutions.

Between the applied performance measures, classification accuracy and geometric mean, there are no major differences.

Table VI
CLASSIFICATION ACCURACIES AND GEOMETRIC MEAN VALUES OF RBFNNS TRAINED USING K-MEANS WITH $N/3$ CLUSTERS IN TOTAL

	Accuracy			G-mean		
Dataset	CS	IO	I	CS	IO	I
Banknote auth.	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
Breast Wisconsin	0.971 ± 0.007	0.971 ± 0.006	$\mathbf{0.972 \pm 0.008}$	0.964 ± 0.008	$\mathbf{0.965 \pm 0.007}$	$\mathbf{0.965 \pm 0.010}$
Dermatology	0.979 ± 0.007	$\mathbf{0.980 \pm 0.006}$	0.979 ± 0.006	0.975 ± 0.008	$\mathbf{0.976 \pm 0.008}$	0.974 ± 0.008
Ecoli	0.868 ± 0.013	$\mathbf{0.870 \pm 0.012}$	$\mathbf{0.870 \pm 0.016}$	0.799 ± 0.020	$\mathbf{0.801 \pm 0.018}$	0.800 ± 0.025
Haberman	0.715 ± 0.023	$\mathbf{0.725 \pm 0.019}$	0.724 ± 0.017	0.472 ± 0.054	$\mathbf{0.489 \pm 0.050}$	0.487 ± 0.045
Heart Cleveland	0.586 ± 0.024	$\mathbf{0.597 \pm 0.020}$	0.590 ± 0.024	$\mathbf{0.181 \pm 0.120}$	0.140 ± 0.121	0.139 ± 0.107
Ionosphere	0.931 ± 0.018	$\mathbf{0.933 \pm 0.017}$	0.931 ± 0.015	0.908 ± 0.025	$\mathbf{0.912 \pm 0.023}$	0.910 ± 0.022
Iris	$\mathbf{0.960 \pm 0.010}$	0.959 ± 0.013	0.959 ± 0.013	$\mathbf{0.958 \pm 0.011}$	0.957 ± 0.013	0.957 ± 0.013
Magic G. Telescope	$\mathbf{0.869 \pm 0.002}$	$\mathbf{0.869 \pm 0.001}$	$\mathbf{0.869 \pm 0.001}$	0.831 ± 0.002	0.831 ± 0.002	$\mathbf{0.832 \pm 0.002}$
Movement libras	$\mathbf{0.795 \pm 0.022}$	0.768 ± 0.024	0.786 ± 0.025	0.578 ± 0.118	0.534 ± 0.199	$\mathbf{0.601 \pm 0.168}$
Musk	$\mathbf{0.875 \pm 0.019}$	0.871 ± 0.019	0.871 ± 0.022	$\mathbf{0.873 \pm 0.019}$	0.869 ± 0.019	0.869 ± 0.022
Parkinsons	$\mathbf{0.885 \pm 0.026}$	0.876 ± 0.029	0.874 ± 0.027	$\mathbf{0.802 \pm 0.040}$	0.778 ± 0.047	0.780 ± 0.039
Pima Ind. Diabetes	0.759 ± 0.013	0.761 ± 0.012	$\mathbf{0.763 \pm 0.012}$	0.697 ± 0.019	0.700 ± 0.016	$\mathbf{0.702 \pm 0.016}$
Seeds	$\mathbf{0.951 \pm 0.010}$	0.950 ± 0.010	0.950 ± 0.007	$\mathbf{0.949 \pm 0.011}$	0.948 ± 0.010	0.948 ± 0.008
Segmentation	0.963 ± 0.004	0.962 ± 0.004	$\mathbf{0.964 \pm 0.004}$	0.962 ± 0.004	0.961 ± 0.005	$\mathbf{0.963 \pm 0.005}$
Sonar all	$\mathbf{0.805 \pm 0.035}$	0.785 ± 0.035	0.782 ± 0.042	$\mathbf{0.799 \pm 0.036}$	0.779 ± 0.035	0.776 ± 0.042
Spectf	0.816 ± 0.021	0.815 ± 0.022	$\mathbf{0.819 \pm 0.024}$	0.644 ± 0.043	$\mathbf{0.654 \pm 0.045}$	$\mathbf{0.654 \pm 0.041}$
Transfusion	0.779 ± 0.011	$\mathbf{0.787 \pm 0.012}$	0.784 ± 0.010	0.546 ± 0.030	$\mathbf{0.559 \pm 0.032}$	0.553 ± 0.025
Vehicle	0.817 ± 0.014	0.816 ± 0.014	$\mathbf{0.818 \pm 0.017}$	0.803 ± 0.016	0.802 ± 0.016	$\mathbf{0.804 \pm 0.020}$
Vertebral column	$\mathbf{0.833 \pm 0.015}$	$\mathbf{0.833 \pm 0.013}$	0.831 ± 0.015	0.773 ± 0.019	$\mathbf{0.774 \pm 0.016}$	0.772 ± 0.019
Vowel context	$\mathbf{0.862 \pm 0.018}$	0.846 ± 0.018	0.855 ± 0.019	$\mathbf{0.830 \pm 0.022}$	0.809 ± 0.024	0.818 ± 0.023
Wine	0.986 ± 0.009	$\mathbf{0.989 \pm 0.008}$	$\mathbf{0.989 \pm 0.009}$	0.986 ± 0.009	$\mathbf{0.989 \pm 0.008}$	$\mathbf{0.989 \pm 0.008}$
Winequality red	$\mathbf{0.609 \pm 0.010}$	$\mathbf{0.609 \pm 0.011}$	0.606 ± 0.010	0.018 ± 0.039	0.014 ± 0.044	$\mathbf{0.024 \pm 0.062}$
Yeast	$\mathbf{0.591 \pm 0.010}$	0.589 ± 0.008	0.588 ± 0.008	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

Table VII
AVERAGE RBF NEURON NUMBERS WHEN MD PSO AND APC-III WERE USED FOR CLUSTERING

	MD PSO			APC-III		
Dataset	CS	IO	I	CS	IO	I
Banknote auth.	26.9 ± 4.7	32.7 ± 1.5	28.3 ± 5.7	525.0 ± 9.9	522.1 ± 10.9	530.4 ± 8.9
Breast Wisconsin	19.5 ± 2.5	18.5 ± 4.8	21.5 ± 0.6	138.3 ± 5.3	139.4 ± 5.0	136.7 ± 5.2
Dermatology	27.1 ± 1.2	17.2 ± 7.2	26.0 ± 4.5	108.1 ± 3.9	106.3 ± 4.3	107.1 ± 4.4
Ecoli	23.0 ± 1.0	19.0 ± 7.0	23.5 ± 1.4	100.9 ± 6.1	78.6 ± 7.9	109.4 ± 7.4
Haberman	4.7 ± 0.8	8.4 ± 4.9	2.7 ± 1.5	96.6 ± 4.1	92.1 ± 4.1	95.3 ± 6.9
Heart Cleveland	17.6 ± 1.2	18.2 ± 3.0	19.0 ± 1.2	106.3 ± 3.6	98.9 ± 2.0	97.7 ± 3.9
Ionosphere	10.9 ± 0.9	16.9 ± 1.2	15.2 ± 4.9	108.1 ± 3.5	81.8 ± 2.4	79.1 ± 2.4
Iris	13.3 ± 1.3	14.1 ± 0.7	13.0 ± 2.3	56.2 ± 3.4	53.5 ± 2.4	54.2 ± 2.7
Letter recognition	96.5 ± 3.8	26.0 ± 0.0	26.0 ± 0.0			
Magic G. Telescope	38.6 ± 25.5	2.0 ± 0.0	2.0 ± 0.0	5271.1 ± 34.5	4383.9 ± 29.6	4174.4 ± 27.2
Movement libras	74.8 ± 0.5	74.8 ± 0.5	74.8 ± 0.5	145.8 ± 2.9	142.4 ± 3.9	144.5 ± 3.5
Musk	15.6 ± 2.8	18.9 ± 2.0	18.8 ± 1.8	151.6 ± 6.2	152.2 ± 5.2	148.1 ± 5.5
Parkinsons	12.8 ± 1.1	13.3 ± 1.0	13.4 ± 0.8	65.6 ± 2.2	65.7 ± 2.5	64.1 ± 2.3
Pima Ind. Diabetes	4.4 ± 0.6	2.7 ± 0.4	2.8 ± 0.4	236.4 ± 7.6	236.5 ± 6.8	238.7 ± 5.0
Seeds	13.4 ± 1.4	11.7 ± 3.7	13.7 ± 1.5	81.2 ± 4.3	79.7 ± 3.7	80.9 ± 3.3
Segmentation	41.8 ± 2.7	29.4 ± 14.7	46.8 ± 2.7	610.0 ± 15.4	558.0 ± 10.0	575.7 ± 10.3
Sonar all	13.2 ± 0.7	13.4 ± 0.7	13.6 ± 0.5	65.3 ± 3.4	65.8 ± 3.1	63.2 ± 2.8
Spectf	15.0 ± 0.7	15.7 ± 0.5	15.6 ± 0.5	62.8 ± 4.7	58.8 ± 3.4	56.5 ± 3.4
Transfusion	19.3 ± 4.3	23.7 ± 3.0	23.5 ± 2.4	189.3 ± 12.0	185.1 ± 12.1	186.4 ± 12.1
Vehicle	24.3 ± 1.6	26.5 ± 1.8	26.8 ± 1.1	292.1 ± 5.5	293.6 ± 6.8	285.6 ± 6.3
Vertebral column	10.4 ± 2.3	16.6 ± 2.0	14.7 ± 2.9	99.4 ± 7.1	92.4 ± 9.5	86.7 ± 9.1
Vowel context	43.1 ± 1.0	41.8 ± 5.1	35.9 ± 7.0	409.0 ± 5.9	398.2 ± 5.7	398.9 ± 6.2
Wine	11.4 ± 1.5	4.9 ± 2.9	14.3 ± 0.6	55.9 ± 2.9	51.5 ± 2.8	52.2 ± 2.7
Winequality red	19.4 ± 2.3	5.0 ± 0.7	9.0 ± 9.5	542.8 ± 12.4	506.8 ± 14.9	545.3 ± 7.4
Yeast	33.6 ± 2.3	20.9 ± 13.4	41.1 ± 7.7	435.2 ± 15.6	410.1 ± 13.8	455.4 ± 7.4

In most cases, the same clustering approach achieves the best performance according to the both measures. The class-specific approach tends to have a slightly better success in comparison with other approaches, when the geometric mean is used as the performance measure. This is also expected as the class-specific approach ensures that also the minority classes will have some clusters assigned to the vicinity of their items. We do assume that the class-specific clustering can be beneficial in classification of imbalanced datasets or if it is important to minimize error on a certain class. However, as mentioned already, the rest of the training process aims at optimizing the overall performance. To really evaluate the clustering approaches in such cases, also the rest of the training process should be modified accordingly.

Table VII reveals that, while the total cluster numbers proposed by different clustering approaches are similar in most cases, for the two largest datasets, Letter recognition and Magic G. Telescope, the input and input-output clustering approaches always propose the smallest possible cluster number (C), when MD PSO is applied. For these datasets, the size of the clustering problem is already so large that these approaches fail to find good solutions with higher cluster numbers with the available particles and iterations and they converge to a local optimum with C centroids. Also for Winequality red, which is one of the largest datasets, the class-specific clustering approach proposes a clearly higher cluster number, while for

Table VIII
ITERATIONS REQUIRED FOR K-MEANS CONVERGENCE WHEN THE TOTAL NUMBER OF CLUSTERS WAS $3C$ AND $\max(5C, \sqrt{N})$

	K-means with $3C$ clusters			K-means with $\max(5C, \sqrt{N})$ clusters		
Dataset	CS	IO	I	CS	IO	I
Banknote auth.	$26.6 \pm 7.8(13.3)$	16.5 ± 7.5	19.8 ± 7.0	$33.4 \pm 8.6(16.7)$	19.9 ± 6.5	18.5 ± 5.3
Breast Wisconsin	$25.2 \pm 7.1(12.6)$	15.7 ± 5.8	18.4 ± 6.7	$23.7 \pm 5.7(11.9)$	13.3 ± 4.1	14.6 ± 3.8
Dermatology	$30.7 \pm 5.0(6.1)$	8.2 ± 2.3	8.9 ± 2.5	$30.1 \pm 5.1(5.0)$	7.7 ± 1.8	7.8 ± 1.7
Ecoli	$31.1 \pm 5.7(6.2)$	10.0 ± 2.8	12.3 ± 4.2	$32.4 \pm 5.4(6.5)$	9.8 ± 3.0	10.4 ± 2.9
Haberman	$14.3 \pm 3.7(7.2)$	9.9 ± 3.3	11.7 ± 4.3	$16.9 \pm 4.5(8.5)$	10.5 ± 3.3	10.8 ± 3.4
Heart Cleveland	$17.1 \pm 3.2(3.4)$	7.9 ± 2.4	8.3 ± 2.7	$18.0 \pm 2.9(3.6)$	7.2 ± 2.0	7.4 ± 2.0
Ionosphere	$16.4 \pm 6.0(8.2)$	10.7 ± 4.6	12.1 ± 5.1	$17.2 \pm 5.4(8.6)$	11.0 ± 4.1	11.8 ± 4.0
Iris	$15.5 \pm 3.5(5.2)$	6.7 ± 2.3	7.5 ± 2.3	$15.0 \pm 3.0(5.0)$	6.5 ± 2.2	7.0 ± 2.0
Letter recognition	$353.4 \pm 31.7(13.6)$	29.1 ± 6.6	72.2 ± 20.6	$393.8 \pm 30.4(15.1)$	33.4 ± 7.2	60.1 ± 17.0
Magic G. Telescope	$69.5 \pm 23.2(34.8)$	38.8 ± 14.4	51.9 ± 27.3	$135.0 \pm 26.5(67.5)$	76.7 ± 19.1	89.6 ± 26.2
Movement libras	$49.6 \pm 5.5(3.3)$	7.1 ± 1.8	7.6 ± 1.9	$48.2 \pm 4.8(3.2)$	6.2 ± 1.4	6.2 ± 1.5
Musk	13.6 ± 5.1 (6.8)	10.2 ± 3.7	10.0 ± 3.8	$15.0 \pm 3.3(7.5)$	10.1 ± 2.9	9.3 ± 2.6
Parkinsons	$14.2 \pm 4.1(7.1)$	9.4 ± 3.8	10.9 ± 4.1	$13.3 \pm 3.4(6.7)$	8.5 ± 2.6	8.8 ± 2.7
Pima Ind. Diabetes	$23.4 \pm 6.6(11.7)$	15.0 ± 5.7	20.6 ± 8.4	$27.0 \pm 7.2(13.5)$	16.4 ± 5.5	17.1 ± 4.4
Seeds	$19.3 \pm 4.6(6.4)$	8.0 ± 2.5	10.2 ± 3.3	$18.4 \pm 3.8(6.1)$	8.1 ± 2.4	8.9 ± 2.4
Segmentation	$67.3 \pm 11.4(9.6)$	15.9 ± 4.8	20.8 ± 6.9	$77.9 \pm 10.1(11.1)$	17.9 ± 4.4	20.5 ± 4.9
Sonar all	$14.2 \pm 4.3(7.1)$	9.2 ± 3.5	8.8 ± 3.0	$12.3 \pm 3.1(6.2)$	7.3 ± 2.4	8.2 ± 2.9
Spectf	$14.1 \pm 4.5(7.1)$	10.8 ± 4.7	11.2 ± 4.1	$13.7 \pm 3.5(6.9)$	9.4 ± 3.0	9.4 ± 2.7
Transfusion	21.6 ± 7.4 (10.8)	15.4 ± 6.7	20.6 ± 8.4	$28.6 \pm 8.3(14.3)$	17.3 ± 6.4	16.4 ± 5.7
Vehicle	$34.4 \pm 7.7(8.6)$	12.9 ± 4.2	19.9 ± 7.7	$41.4 \pm 8.6(10.4)$	13.7 ± 3.6	16.9 ± 5.2
Vertebral column	$21.4 \pm 4.5(7.1)$	10.1 ± 3.5	13.5 ± 4.7	$23.7 \pm 5.6(7.9)$	10.6 ± 3.4	12.3 ± 4.6
Vowel context	$54.6 \pm 7.9(6.1)$	11.4 ± 3.3	17.0 ± 5.3	$56.4 \pm 7.7(6.3)$	10.7 ± 3.2	13.9 ± 3.4
Wine	$16.6 \pm 3.5(5.5)$	6.9 ± 2.1	7.9 ± 2.6	$16.1 \pm 3.1(5.4)$	6.5 ± 1.8	7.7 ± 2.3
Winequality red	$42.3 \pm 9.6(10.6)$	18.4 ± 6.2	26.5 ± 9.1	$48.5 \pm 8.1(12.1)$	19.9 ± 5.1	20.9 ± 6.5
Yeast	$67.0 \pm 10.3(7.4)$	20.9 ± 6.5	26.7 ± 7.8	$75.1 \pm 11.4(8.3)$	19.8 ± 4.9	22.3 ± 6.3

the smaller datasets such a behavior is not observed. The input-output clustering produces the lowest cluster numbers on several datasets (Dermatology, Ecoli, Segmentation, Wine, Winequality red). On most of these datasets the performance obtained by the approach is worse than with other approaches and the weight of the output part is high. This supports the assumption that overly weighted output part creates sub-optimal local optima on lower dimensions (as the items from the same class are attracted to each other due to the shared output part) and, therefore, the input-output clustering approach can harm the performance of the RBFNNs in such cases.

When APC-III is applied, the cluster numbers are quite similar for all the clustering approaches. The class-specific approach often leads to a slightly higher cluster number. This is also expected as APC-III always divides the data space into clusters with a fixed radius. For the class-specific clustering approach some overlap of clusters is probable as the classes are handled separately.

Table VIII shows that without any exceptions K-means converges with least iterations for (a single clustering of) the class-specific clustering approach. As discussed in Section IV-A, the class-specific approach would be C times faster than the input clustering, if the iteration number for a single clustering would be the same. As the iteration number is even lower, the computational benefit of using the class-specific approach is larger.

D. Statistical Evaluation of the Results

To evaluate whether the differences observed between the compared clustering approaches are statistically significant, we first perform the Friedman test to control the family-wise error. It is a rank-based non-parametric test to evaluate, whether the null-hypothesis can be rejected and the results show some statistically significant differences. A detailed description of the test can be found in [59]. We applied the test separately for each clustering algorithm and performance measure. The obtained ρ-values are given in Table IX. The null-hypothesis can be rejected at 0.05 significance level if the ρ-value is below 0.05. Such values are bolded in the table. The results of the Friedman test confirm the observations made above. There are no statistically significant differences between the clustering approaches, when APC-III and K-means with $N/3$ clusters are applied, but in the other cases differences are significant.

Table IX
FRIEDMAN TEST ρ-VALUES FOR COMPARISONS PERFORMED WITH DIFFERENT CLUSTERING ALGORITHMS AND PERFORMANCE MEASURES

Algorithm	ρ (Acc.)	ρ (G-mean)
MD PSO	$\mathbf{0.0057}$	$\mathbf{0.0087}$
APC-III	0.9575	0.1699
K-means ($3C$)	$\mathbf{0.000018}$	$\mathbf{0.0000018}$
K-means ($\max(5C, \sqrt{N})$)	$\mathbf{0.0207}$	$\mathbf{0.0147}$
K-means ($N/3$)	0.8721	0.5682

Next we perform the Wilcoxon Signed-Ranks test for those cases in which statistically significant differences were detected to evaluate the significance of the pair-wise differences. The Wilcoxon Signed-Ranks test ranks the differences in performances of two classifiers for each dataset, ignoring the signs, and compares the ranks for the positive and the negative differences. The ranks are used to compute a T-value as described e.g. in [59] and this value is compared to a critical value to evaluate whether the difference is statistically significant. For 25 datasets, the null hypothesis can be rejected at the 0.05 significance level if T is less than 89 and at the 0.01 significance level if T is less than 68. The obtained T-values are given in Table X. In each case the better performing approach is bolded and the corresponding T-value is bolded if it signals that the null hypothesis can be rejected at the 0.05 significance level.

The results confirm that when using MD PSO, K-means with $3C$ clusters and K-means with $\max(5C, \sqrt{N})$ clusters

Table X
WILCOXON SIGNED-RANKS TEST T VALUES FOR COMPARISONS PERFORMED WITH DIFFERENT CLUSTERING ALGORITHMS AND PERFORMANCE MEASURES

Approach 1	T (Acc.)	T (G-mean)	Approach 2
MD-PSO			
CS	**34**	**46,5**	IO
CS	**57**	**68,5**	I
IO	141		**I**
IO		154,5	I
K-means with $3C$ clusters			
CS	**25**	**17.5**	IO
CS	**27**	**9.5**	I
IO	143		I
IO		146.5	**I**
K-means with $\max(5C, \sqrt{N})$ clusters			
CS	**56**	**55.5**	IO
CS	**64**	**49.5**	I
IO	129	144,5	**I**

the class-specific clustering approach produces significantly better results than other clustering approaches according to the both applied performance measures. Between the input and input-output clustering approaches no statistically significant differences are detected.

VI. CONCLUSIONS

In this paper, we proposed using class-specific clustering approach in RBFNN training. We demonstrated that it can achieve the best performance in terms of both classification accuracy and computational efficiency in comparison to traditional clustering approaches. This is in accordance with the success of the divide-and-conquer paradigm over several other data analysis problems. In this case, depending on the dataset, clustering the whole dataset at once can be a complicated task due to the occurrence of local optima in the solution space. Therefore, performing the class-specific clustering approach can help in finding a better solution. Furthermore, the proposed class-specific clustering approach provides a simple way to exploit class-information and make each cluster centroid correspond to a cluster of items belonging to a single class.

For an extensive and detailed comparison of the class-specific approach with the input and input-output clustering approaches, we first analyzed their computational complexities and concluded that the class-specific approach can reduce the overall complexity even for the simplest clustering methods as it reduces both the number of items to be clustered at once and the number of clusters to be found. The only cost is that the clustering must be repeated separately for each class, but this cost is compensated by the reduced complexity of the clustering process. If a more complicated clustering algorithm is applied, the computational gain obtained by using the class-specific clustering approach increases proportional with the the complexity of the algorithm. Therefore, the class-specific clustering approach should be favored even without improvements in the classification performance. However, an extensive set of experiments on 25 benchmark datasets demonstrated that the class-specific clustering approach can also produce better classification results than the other clustering approaches. One of the applied algorithms was a dynamic clustering method using MD PSO, which is an evolutionary optimization algorithm. We are not aware of any earlier proposed combination of an evolutionary optimization method and the class-specific clustering approach, while the approach is especially useful for evolutionary methods due to their higher computational complexity.

Our results and statistical evaluations show that, when the RBF center number is kept relatively low (up to the square root of the number of training items), the class-specific clustering can indeed produce centroids that lead to significant improvements in the final classification accuracy compared to the other approaches. For larger numbers of RBF centers no statistically significant differences between the approaches were observed. On the other hand, for these higher cluster numbers the computational gain obtained by using the class-specific clustering approach becomes even more significant.

Our results further show that there were no significant differences between the input and input-output clustering approaches. However, we observed that the input-output clustering approach can be beneficial when the number of classes is low in comparison with the feature vector dimension. When the number of classes is high, the results suffer from the use of the input-output clustering approach.

In this study, we did not apply any post-processing on the clustering results and the RBF neurons were directly placed at the cluster centroids even for the proposed class-specific approach when two different classes have completely overlapping centroids. Such a case obviously indicates that the area should be modeled with more RBF centers, but we have not adapted the RBFNN topology on purpose for a fair comparison and left it as future work. Similarly, the rest of training process was now kept simple, but will be improved to the level of the state-of-the-art in the future. Finally, the class-specific clustering approach can be very useful when working with imbalanced datasets or in cases where a particular class is crucial to classify correctly. These will be topics for our future work.

REFERENCES

[1] A. Alexandridis and E. Chondrodima, "A medical diagnostic tool based on radial basis function classifiers and evolutionary simulated annealing," *J. of Biomedical Informatics*, vol. 49, pp. 61 – 72, 2014.

[2] R.-J. Lian, "Adaptive self-organizing fuzzy sliding-mode radial basis-function neural-network controller for robotic systems," *IEEE Trans. Ind. Electron.*, vol. 61, no. 3, pp. 1493–1503, Mar. 2014.

[3] H. Yu, T. Xie, S. Paszczynski, and B. Wilamowski, "Advantages of radial basis function networks for dynamic system design," *IEEE Trans. Ind. Electron.*, vol. 58, no. 12, pp. 5438–5450, Dec. 2011.

[4] W. Shen, X. Guo, C. Wu, and D. Wu, "Forecasting stock indices using radial basis function neural networks optimized by artificial fish swarm algorithm," *Knowledge-Based Syst.*, vol. 24, no. 3, pp. 378 – 385, 2011.

[5] Y.-S. Hwang and S.-Y. Bang, "An efficient method to construct a radial basis function neural network classifier," *Neural Networks*, vol. 10, no. 8, pp. 1495 – 1503, 1997.

[6] M. J. Er, S. Wu, J. Lu, and H. L. Toh, "Face recognition with radial basis function (rbf) neural networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 3, pp. 697–710, May 2002.

[7] D. Wettschereck and T. G. Dietterich, "Improving the performance of radial basis function networks by learning center locations," in *Advances in Neural Inform. Process. Syst. 4*, 1992, pp. 1133–1140.

[8] S. A. Billings and G. L. Zheng, "Radial basis function network configuration using genetic algorithms," *Neural Networks*, vol. 8, no. 6, pp. 877 – 890, 1995.

[9] P. H. Fidencio, R. J. Poppi, and J. C. de Andrade, "Determination of organic matter in soils using radial basis function networks and near infrared spectroscopy," *Analytica Chimica Acta*, vol. 453, no. 1, pp. 125 – 134, 2002.

[10] F. Schwenker, H. A. Kestler, and G. Palm, "Three learning phases for radial-basis-function networks," *Neural Networks*, vol. 14, no. 45, pp. 439 – 458, 2001.

[11] R. Neruda and P. Kudov, "Learning methods for radial basis function networks," *Future Generation Comput. Syst.*, vol. 21, no. 7, pp. 1131 – 1142, 2005.

[12] T. Poggio and F. Girosi, "A theory of networks for approximation and learning," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, USA, A.I. Memo No. 1140, 1989.

[13] ——, "Networks for approximation and learning," *Proc. IEEE*, vol. 78, no. 9, pp. 1481–1497, Sep. 1990.

[14] J.-X. Peng, K. Li, and D.-S. Huang, "A hybrid forward algorithm for rbf neural network construction," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1439–1451, Nov 2006.

[15] S. Haykin, *Neural Networks - A Comprehensive Foundation, 2nd ed.* Prentice Hall International, Inc., 1999.

[16] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281 – 294, 1989.

[17] F. Bauer and M. A. Lukas, "Comparing parameter choice methods for regularization of ill-posed problems," *Mathematics and Computers in Simulation*, vol. 81, no. 9, pp. 1795 – 1841, 2011.

[18] N. Karayiannis and M. Randolph-Gips, "On the construction and training of reformulated radial basis function neural networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 835–846, July 2003.

[19] J. Gonzalez, I. Rojas, J. Ortega, H. Pomares, F. Fernandez, and A. Diaz, "Multiobjective evolutionary optimization of the size, shape, and position parameters of radial basis function networks for function approximation," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1478–1495, Nov 2003.

[20] C. Harpham, C. Dawson, and M. Brown, "A review of genetic algorithms applied to training radial basis function networks," *Neural Computing & Appl.*, vol. 13, no. 3, pp. 193–201, 2004.

[21] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411–1423, Nov 2006.

[22] G. Babu and S. Suresh, "Sequential projection-based metacognitive learning in a radial basis function network for classification problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 2, pp. 194–206, Feb. 2013.

[23] S. Suresh, N. Sundararajan, and P. Saratchandran, "A sequential multi-category classifier using radial basis function networks," *Neurocomputing*, vol. 71, no. 79, pp. 1345 – 1358, 2008, 15th Eur. Symp. Artificial Neural Networks.

[24] N. Benoudjit and M. Verleysen, "On the kernel widths in radial-basis function networks," *Neural Process. Lett.*, vol. 18, no. 2, pp. 139–154, 2003.

[25] T. Xie, H. Yu, J. Hewlett, P. Rozycki, and B. Wilamowski, "Fast and efficient second-order method for training radial basis function networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 4, pp. 609–619, Apr. 2012.

[26] P. C. Pendharkar, "A hybrid radial basis function and data envelopment analysis neural network for classification," *Computers and Operations Research*, vol. 38, no. 1, pp. 256 – 266, 2011.

[27] S.-K. Oh, W.-D. Kim, W. Pedrycz, and S.-C. Joo, "Design of K-means clustering-based polynomial radial basis function neural networks (pRBF NNs) realized with the aid of particle swarm optimization and differential evolution," *Neurocomputing*, vol. 78, no. 1, pp. 121 – 132, 2012.

[28] B. Fornberg and C. Piret, "On choosing a radial basis function and a shape parameter when solving a convective PDE on a sphere," *J. of Computational Physics*, vol. 227, no. 5, pp. 2758 – 2780, 2008.

[29] J. Haddadnia, K. Faez, and M. Ahmadi, "A fuzzy hybrid learning algorithm for radial basis function neural network with application in human face recognition," *Pattern Recognition*, vol. 36, no. 5, pp. 1187 – 1202, 2003.

[30] A. D. Niros and G. E. Tsekouras, "A novel training algorithm for {RBF} neural network using a hybrid fuzzy clustering approach," *Fuzzy Sets and Systems*, vol. 193, pp. 62 – 84, 2012, theme : Data Analysis.

[31] C.-L. Chen, W.-C. Chen, and F.-Y. Chang, "Hybrid learning algorithm for Gaussian potential function networks," *Control Theory and Appl., IEE Proc. D*, vol. 140, no. 6, pp. 442–448, Nov. 1993.

[32] Z. Uykan, C. Guzelis, M. Celebi, and H. Koivo, "Analysis of input-output clustering for determining centers of RBFN," *IEEE Trans. Neural Netw.*, vol. 11, no. 4, pp. 851–858, Jul. 2000.

[33] Z. Uykan and H. Koivo, "Analysis of augmented-input-layer RBFNN," *IEEE Trans. Neural Netw.*, vol. 16, no. 2, pp. 364–369, Mar. 2005.

[34] G. E. Tsekouras and J. Tsimikas, "On training RBF neural networks using input-output fuzzy clustering and particle swarm optimization," *Fuzzy Sets and Syst.*, vol. 221, pp. 65 – 89, 2013.

[35] M. Han and J. Xi, "Efficient clustering of radial basis perceptron neural network for pattern recognition," *Pattern Recognition*, vol. 37, no. 10, pp. 2059 – 2067, 2004.

[36] W. Pedrycz, "Conditional fuzzy clustering in the design of radial basis function neural networks," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 601–612, Jul. 1998.

[37] A. Staiano, R. Tagliaferri, and W. Pedrycz, "Improving RBF networks performance in regression tasks by means of a supervised fuzzy clustering," *Neurocomputing*, vol. 69, no. 1315, pp. 1570 – 1581, 2006.

[38] K. Koutroumbas and A. Pouliakis, "A novel algorithm for training rbf networks," in *Proc. 2nd Hellenic Conf. on AI*, Thessaloniki, Greece, Apr. 2002, pp. 283–292.

[39] Y.-S. Hwang and S.-Y. Bang, "A neural network model APC-III and its application to unconstrained handwritten digit recognition," in *Proc. Int. Conf. on Neural Inform. Process.*, 1994, pp. 1500–1505.

[40] ——, "An efficient method to construct a radial basis function neural network classifier and its application to unconstrained handwritten digit recognition," in *Proc. 13th Int. Conf. on Pattern Recognition*, vol. 4, Aug. 1996, pp. 640–644 vol.4.

[41] Y.-J. Oyang, S.-C. Hwang, Y.-Y. Ou, C.-Y. Chen, and Z.-W. Chen, "Data classification with radial basis function networks based on a novel kernel density estimation algorithm," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 225–236, Jan. 2005.

[42] D. G. S. Richard O. Duda, Peter E. Hart, *Pattern Classification, 2nd ed.* John Wiley & Sons, Inc., 2000.

[43] D. Arthur, B. Manthey, and H. Roglin, "k-means has polynomial smoothed complexity," in *Proc. 50th Annu. IEEE Symp. Found. Comput. Sci.*, Oct 2009, pp. 405–414.

[44] J. Stastny and V. Skorpil, "Analysis of algorithms for radial basis function neural network," in *Personal Wireless Communications*, R. Bestak, B. Simak, and E. Kozlowska, Eds. Springer US, 2007, vol. 245, pp. 54–62.

[45] A. Idri, A. Abran, and S. Mbarki, "An experiment on the design of radial basis function neural networks for software cost estimation," in *Proc. 2nd Conf. Information and Communication Technologies*, vol. 1, 2006, pp. 1612–1617.

[46] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Fractional particle swarm optimization in multidimensional search space," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 2, pp. 298–319, 2010.

[47] T. Ince, S. Kiranyaz, and M. Gabbouj, "Evolutionary RBF classifier for polarimetric SAR images," *Expert Syst. with Appl.*, vol. 39, no. 5, pp. 4710 – 4717, 2012.

[48] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks*, Perth, Australia, 1995, pp. 1942–1948.

[49] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Congr. Evolutionary Computation*, Anchorage, Alaska, USA, 1998, pp. 69–73.

[50] S. Kiranyaz, T. Ince, and M. Gabbouj, *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Springer, 2014.

[51] M. Omran, A. Salman, and A. Engelbrecht, "Image classification using particle swarm optimization," in *Proc. 4th Asia-Pacific Conf. Simulated Evolution and Learning*, Singapore, 2002, pp. 370–374.

[52] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proc. American Math. Soc.*, vol. 7, no. 1, Feb. 1956.

[53] L. Xu, "Bayesian Ying-Yang machine, clustering and number of clusters," *Pattern Recognition Lett.*, vol. 18, no. 11-13, pp. 1167 – 1178, 1997.

[54] B. S. Anatolyevich, "ALGLIB numerical analysis and data processing library," 2015. [Online]. Available: http://www.alglib.net/

[55] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Trans. Math. Softw.*, vol. 8, no. 1, pp. 43–71, Mar. 1982.

[56] Y. Sun, M. Kamel, and Y. Wang, "Boosting for learning multiple classes with imbalanced class distribution," in *Proc. 6th Int. Conf. Data Mining*, Dec 2006, pp. 592–602.

[57] "Tampere center for scientific computing," 2015. [Online]. Available: http://www.tut.fi/en/tampere-center-for-scientific-computing/index.htm

[58] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[59] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006.

Publication VIII

Jenni Raitoharju, Kaveh Samiee, Serkan Kiranyaz, Moncef Gabbouj, "Particle Swarm Clustering Fitness Evaluation with Computational Centroids," *Submitted to Swarm and Evolutionary Computation*, Aug. 2016.

Particle Swarm Clustering Fitness Evaluation with Computational Centroids

Jenni Raitoharju[a,*], Kaveh Samiee[a], Serkan Kiranyaz[a,1], Moncef Gabbouj[a]

[a] *Tampere University of Technology, Department of Signal Processing, P.O.Box 553, FI-33101 Tampere, Finland*

Abstract

In this paper, we propose a new way to carry out fitness evaluation in dynamic Particle Swarm Clustering (PSC) with centroid-based encoding. Generally, the PSC fitness function is selected among the clustering validity indices and most of them directly depend on the cluster centroids. In the traditional fitness evaluation approach, the cluster centroids are replaced by the centroids proposed by a particle position. We propose to first compute the centroids of the corresponding clusters and then use these computational centroids in fitness evaluation. The proposed way is called Fitness Evaluation with Computational Centroids (FECC). We conducted an extensive set of comparative evaluations against the traditional approach. The results show that FECC leads to a clear improvement in clustering results with the most of the fitness functions considered in this study. The proposed approach was found especially beneficial when underclustering is a problem. Furthermore, we evaluated 31 fitness functions based on 17 clustering validity indices using two PSC methods over a large number of synthetic and real data sets with varying properties. We used three different partition similarity measures to compare the clustering results against the ground-truth data. The top three fitness functions were Xu index, WB index, and Dunn variant DU_{23} applied using FECC. These indices were consistently performing well for both PSC

*Corresponding author

Email addresses: jenni.raitoharju@tut.fi (Jenni Raitoharju), kaveh.samiee@gmail.com (Kaveh Samiee), serkan.kiranyaz@tut.fi (Serkan Kiranyaz), moncef.gabbouj@tut.fi (Moncef Gabbouj)

[1]Present address: Department of Electrical Engineering, College of Engineering, Qatar University, Qatar

methods, for all data distributions, and according to all the partition similarity measures. Further guidance for improved fitness function selection in different situations is provided in the paper.

Keywords: Particle swarm optimization, Pattern clustering, Validity index, Swarm intelligence

---

1. Introduction

Particle Swarm Optimization (PSO) has global convergence ability and, due to its stochastic nature, it can avoid local optima. Therefore, it has been widely exploited to solve complex clustering tasks, where simpler clustering algorithms, such as K-means, are likely to get stuck into a local optimum – possibly far from a satisfactory result. PSO can be also used to simultaneously optimize the number of clusters. It is known that proper fitness evaluation is critical to obtain successful clustering solutions. However, the Particle Swarm Clustering (PSC) fitness evaluation has not received much attention before. Most novel PSC applications simply adopt their fitness evaluation approach from a previous work, while in most cases, the results could be easily improved via minor modifications in the fitness evaluation as will be shown in this paper.

We propose a novel approach to compute the fitness of a particle position. We call the approach Fitness Evaluation with Computational Centroids (FECC). It is usable with the most clustering validity index functions (CVIs), each of which can generally be used as the fitness function for PSC. The idea behind FECC is to replace the particle position in the fitness function with the true computational centroids of the corresponding clusters. We conduct a thorough comparison of FECC and the traditional fitness evaluation with several different CVIs to show the superiority of the new approach.

Our second objective is to exhaustively investigate which CVIs are the most suitable fitness functions for PSC. Despite the importance of this question, exhaustive comparisons of PSC fitness functions are lacking. While several general CVI comparisons can be found in the literature, the results cannot be directly assumed to help in PSC fitness function selection due to different requirements. In this paper, we will discuss previous CVI comparisons and the results relevant for PSC fitness function selection. Thereafter,

we conduct an extensive comparison of PSC fitness functions based on different CVIs.

As different fitness functions or a different approach to perform fitness evaluation may be optimal for different data types, we will evaluate the effect of typical dataset characteristics (true number of clusters, dimensionality, asymmetric density, overlap, and noise). The results will be analyzed with statistical significance tests. In the experiments, we use two different PSC approaches, namely Multi-elitist Particle Swarm Optimization (MEPSO) [1] and Multi-dimensional Particle Swarm Optimization (MD-PSO) along with the Fractional Global Best Formation (FGBF) method [2]. Both approaches use centroid-based particle encoding and can be used for dynamic clustering, where the optimal number of clusters is searched for simultaneously with the optimal centroid positions.

The rest of the paper is organized as follows. PSC and the applied PSC methods, MEPSO and MD-PSO, are briefly described in Section 2. Fitness evaluation in PSC is discussed in Section 3. First, the importance and problems of selecting a proper CVI as the fitness function are discussed in Subsection 3.1 and the proposed FECC approach is introduced in Subsection 3.2. Section 4 introduces a rich set of CVIs considered in this paper along with previous CVI comparisons. Experimental results are given in Section 5 and, finally, Section 6 concludes the paper.

2. Particle Swarm Clustering

The basic form of the PSO algorithm was introduced in [3] and later modified in [4]. In the algorithm a swarm of S particles flies stochastically through an N-dimensional search space where each particle's position represents a potential solution to an optimization problem. Each particle p with current position $\mathbf{x}_p$ and current velocity $\mathbf{v}_p$ remembers its personal best solution so far, $\mathbf{b}_p$. The swarm as a whole remembers the overall best solution globally achieved so far, $\mathbf{b}_S$. The particles experience attraction toward the best solutions and, after some time, the swarm typically converges to an optimum.

Due to its stochastic nature, PSO can avoid some local optima. However, for the basic form of the PSO algorithm, premature convergence to a local optimum is a common problem and, therefore, several modifications or extensions of the basic form have been introduced [5], such as the Perturbed PSO [6], Orthogonal Learning PSO [7], or different local neighborhood topologies

[8], e.g. the Fully Informed PSO [9]. In this paper, we will concentrate on two different PSO modifications, which can avoid premature convergence to some extent, namely MEPSO (Section 2.1) and MD-PSO with FGBF (Section 2.2).

In clustering, similar to other PSO applications, each particle's position should represent a potential solution to the problem. Most often this is realized by encoding the position of particle p as $\mathbf{x}_p = \{\mathbf{m}_{p,1}, \ldots, \mathbf{m}_{p,j}, \ldots, \mathbf{m}_{p,K}\}$, where $\mathbf{m}_{p,j}$ represents the j^{th} (potential) cluster centroid in an N-dimensional data space and K is the number of clusters. Each element of the K-dimensional position of the particle $\mathbf{x}_p$ is now an N-dimensional position in the data space. Also different particle encodings have been proposed such as partition-based encoding [10], where each particle is a vector of n integers, where n is the number of data items to be clustered and the i^{th} element represents the cluster label assigned to item $i, i \in \{1, .., n\}$, but only the centroid-based encoding will be further discussed in this article.

The first centroid-based PSC technique was introduced in [11]. The major limitation of the proposed method was the need to manually define the number of clusters, K, *a priori*. Another clustering technique proposed in [12] overcame this limitation by using binary PSO to select which of the potential particle centroids should be included in the final solution, but in this technique the K-means algorithm was used to refine centroid positions.

Das et al. proposed in [1] an algorithm using MEPSO for both cluster number selection and the actual clustering. They proposed a particle encoding scheme, where each particle has a user-defined maximum number of cluster centroids along with activation thresholds for each centroid. The activation thresholds were optimized simultaneously to optimizing the centroid positions and they defined which centroids were included in the actual solutions represented by the particle. The same particle encoding was adopted in [13, 14] for Differential Evolution Particle Swarm Optimization (DEPSO). A different particle encoding allowing optimization of K was proposed in [2]. Each particle has a position in every solution space dimensionality from $K_{\min}$ to $K_{\max}$ and the proposed MD-PSO algorithm is able to move particles from one dimensionality to another. More PSO variants applied for clustering are reviewed in [15, 16].

2.1. Clustering with Multi-Elitist Particle Swarm Optimization

In MEPSO [1], each particle has a growth rate β_p. If the fitness value of particle p at iteration t is better than at iteration $(t-1)$, β_p is increased.

At iteration t, after the particle fitness evaluations have been performed, all the particles having a better local best value, $\mathbf{b}_p(t)$, than the previous global best, $\mathbf{b}_S(t-1)$, are moved into a candidate area. The local best having the highest growth rate is assigned as the new global best, $\mathbf{b}_S(t)$. Thus, the new global best may not have the highest fitness value, but it has improved more during the last iteration. This can prevent the swarm from gathering too close to a suboptimal global best solution in an early phase of the search process. The pseudo-code of MEPSO can be found in [1].

The particle encoding used for MEPSO clustering was also proposed in [1]. Given a user defined maximum number of clusters, $K_{\max}$, the position of particle p is encoded as a $K_{\max} + K_{\max} * N$ vector $\mathbf{x}_p = \{T_{p,1}, \ldots, T_{p,K_{\max}}, \mathbf{m}_{p,1}, \ldots, \mathbf{m}_{p,K_{\max}}\}$, where $T_{p,j}$, $j \in \{1, \ldots, K_{\max}\}$ is an activation threshold in the range of $[0, 1]$ and $\mathbf{m}_{p,j}$ represents the j^{th} (potential) cluster centroid. If $T_{p,j} > 0.5$, the corresponding j^{th} centroid is included in the solution. Otherwise, the cluster defined by the j^{th} centroid is inactive. The minimum cluster number is defined to be two. If there are less than two active clusters in a solution, one or two randomly selected activation thresholds, $T_{p,j} < 0.5$, are re-initialized to a random value in the range of $]0.5, 1]$. In our implementation, similar to [13], we also deactivate any cluster having less than two items by setting its activation threshold to a random value in the range of $[0, 0.5]$ and then check whether the condition on the minimum number of clusters is still satisfied.

The problem with this kind of particle encoding is that basically each particle position represents partitions with all the considered numbers of clusters. Most likely the algorithm will first encounter some decent solutions with smaller numbers of clusters. The purpose of the algorithm is to gradually guide the particles toward better solutions. However, the same cluster positions are often not ideal for different numbers of clusters as illustrated in Fig. 1. If the algorithm first finds a near-optimal solution with two clusters, just adding a third cluster centroid will not be a good solution with three clusters. Therefore, this particle encoding creates strong local optima with lower numbers of clusters and, indeed, underclustering (i.e. finding partitions with too few clusters) is a common problem.

Originally, MEPSO was proposed with the kernelized CS CVI, but we found this index computationally too expensive for larger datasets and, therefore, it was excluded from this paper.

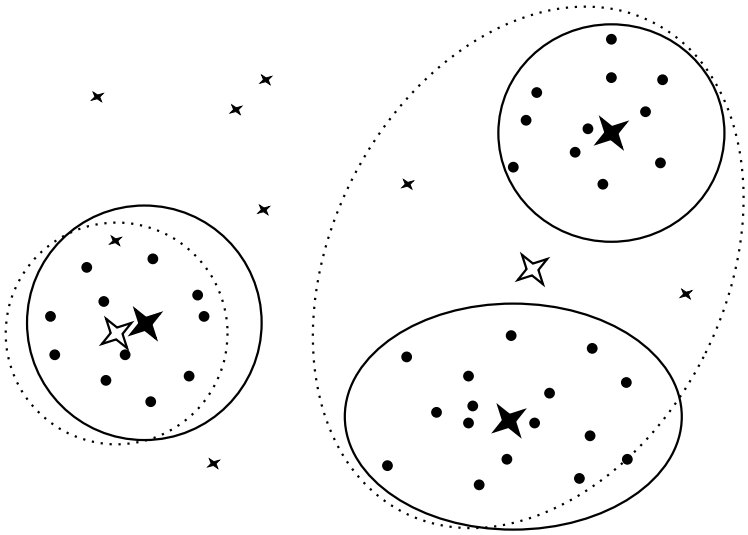


Figure 1: Possible cluster centroids (stars) for a dataset. Filled large stars form a desired solution with three clusters and unfilled large stars with two clusters.

2.2. Clustering with Multi-dimensional Particle Swarm Optimization

MD-PSO [2, 17] is an extension of the basic PSO algorithm, where particles can search for solutions with different dimensionalities within a given dimensionality range, $\{d_{\min}, \ldots, d_{\max}\}$. In order to accomplish this, each particle has two sets of components, each of which has been subjected to one of the two independent and consecutive PSO processes. The first one is the regular positional PSO, which takes place in each particle's current dimensionality, d_p, and the second one is a dimensional PSO, which allows particles to move among search spaces in different dimensions. For each dimensionality d, the swarm keeps track of the global best position so far achieved, $\mathbf{b}_S^d$, and each particle p keeps track of its last position $\mathbf{x}_p^d$, velocity $\mathbf{v}_p^d$ and personal best position $\mathbf{b}_p^d$ in that particular dimensionality d so that when it re-visits the same dimensionality at a later time, it can perform the regular positional updates using this information. The dimensional PSO process may then again move the particle to another dimensionality, where it will remember its positional status and will be updated through the positional PSO process, and so on. Similarly to the positional PSO, the dimensional PSO process uses the personal best dimensionality (in which the personal best fitness score has so far been achieved) of each particle, db_p, and the global best dimensionality, db_S, to attract the particles toward a better dimensional solution. Finally, the global best solution in dimensionality db_S, $\mathbf{b}_S^{db_S}$, represents the optimal solution and dimensionality, respectively.

Both the basic PSO algorithm and its multi-dimensional extension MD-PSO suffer from premature convergence to a local optimum particularly when a high dimensional optimization problem with a multi-modal fitness surface is encountered. The premature convergence is mainly caused by a loss of diversity i.e. the particles gather too close to $\mathbf{b}_S$ in an early phase and lose their ability to explore new potential solutions. FGBF [2] that is a plug-in to the (MD-)PSO process can efficiently address the premature convergence problem. The main idea of FGBF is to create at every iteration an additional artificial solution $\mathbf{b}_A$ by combining the best individual elements of particles' solutions evaluated by fractional fitness scores, $f(p, d, j)$. This artificial solution is then compared to $\mathbf{b}_S$ and, if it turns out to have a higher fitness value, $\mathbf{b}_A$ will replace $\mathbf{b}_S$ as the global best. If, on the other hand, the new solution $\mathbf{b}_A$ is not better than $\mathbf{b}_S$, the latter will be used in the subsequent iteration. When FGBF is used in combination with MD-PSO a separate artificial solution, $\mathbf{b}_A^d$, is created for every dimensionality d within the dimensionality range $\{d_{\min}, \ldots, d_{\max}\}$ and in every dimensionality $\mathbf{b}_A^d$ solution

competes with $\mathbf{b}_S^d$ solution. However, depending on the optimization task, during the formation of $\mathbf{b}_A^d$ it may be possible to combine elements from particle positions with different dimensionalities (In the clustering case, this means that cluster centroids can be collected from solutions with different K).

MD-PSO and FGBF were first applied to clustering in [2] and the method was also exploited, for example, in [18, 19, 20], where it has been shown that MD-PSO with FGBF can achieve an optimal clustering performance even in highly complicated datasets. The particle encoding used is a straightforward multi-dimensional extension of the centroid-based encoding used in [11], i.e. $\mathbf{x}_p^d = \{\mathbf{m}_{p,1}, \ldots, \mathbf{m}_{p,j}, \ldots, \mathbf{m}_{p,d}\}$. Now each particle has a position in every dimensionality $d \in \{d_{\min}, \ldots, d_{\max}\}$ and in the d^{th} dimensionality particle positions represent d potential cluster centroids. The dimensional PSO process leads particle towards solutions with a better number of clusters while the interleaved positional PSO process helps find better solutions with a certain number of clusters. Having separate solutions for each cluster number, this particle encoding is not creating similar strong local optima with low numbers of clusters as the particle encoding used with MEPSO (described in Section 2.1) and underclustering is not a problem to the same extent.

The objective of the FGBF process is to combine the best individual elements (cluster centroids in this case) to create artificial solutions $\mathbf{b}_A^d$ to compete with $\mathbf{b}_S^d$ solutions in every dimensionality $d \in \{d_{\min}, \ldots, d_{\max}\}$. If among all possible cluster centroids represented by stars in Fig. 1, one could combine the unfilled large stars when creating $\mathbf{b}_A^2$ and the filled large stars when creating $\mathbf{b}_A^3$, these solutions could help the PSO process converge to a better overall clustering solution. However, finding these desired cluster centroid combinations among all the possibilities is not simple. It is not feasible to consider all the possible centroid combinations, but if the quality of a certain centroid is evaluated only locally with respect to the data items without considering other centroids, undesired solutions are likely to be obtained. Therefore, the FGBF operation exploits Minimum Spanning Trees (MSTs) [21] to ensure that the selected centroids represent different natural clusters. The pseudo-codes and further details of MD-PSO, FGBF, and their application to clustering can be found in [2, 17].

Originally in [2], the fractional fitness scores for centroids were evaluated based on their distances to the data items clustered into the corresponding cluster in the PSO position from which the centroid has been taken. The main reason to use such approach was that the fractional fitness scores could

thus be calculated simultaneously to computing the CVI used as a fitness function (Eq. (2)). Naturally it was not guaranteed that the same data items would be assigned to the cluster defined by the centroid when combined with different centroids in solution $\mathbf{b}_A^d$. Nevertheless, this approach gave a functioning approximation of the centroid quality. In this paper, we use a slightly different fractional fitness score, $f(p, d, j)$, to approximate the quality of the centroid defined by the j^{th} element of particle p's position in dimensionality d, $\mathbf{x}_{p,j}^d = \mathbf{m}_{p,j}$

$$f(p, d, j) = \sum_{i=1}^{M} \frac{\|\mathbf{x}_{[i]} - \mathbf{m}_{p,j}\|}{M} \tag{1}$$

where $\mathbf{x}_{[i]}$ represents the i^{th} closest data item to $\mathbf{m}_{p,j}$. Due to computational simplicity, only data items assigned to cluster C_j defined by $\mathbf{m}_{p,j}$ in solution $\mathbf{x}_p$ are considered. $M = \min(10, n_j)$, where n_j is the number of data items assigned to cluster C_j.

3. Fitness Evaluation in Particle Swarm Clustering

3.1. Fitness Function Selection

The quality of a partition produced by a clustering method can be evaluated with three types of CVIs: external, internal, and relative. External CVIs compare clustering results with the ground truth information. As PSC is usually exploited in situations, where no such information is available, external CVIs are cannot be used as fitness functions. Instead, they provide a means to evaluate the performance achieved using internal and relative CVIs as fitness functions in test circumstances where the optimal partition is known. Internal CVIs evaluate clusters solely based on the data itself. Commonly used measures are, for example, cluster compactness and separation. Relative CVIs can be used to compare two clustering results and to indicate which one is better. While most external CVIs are relative, the term usually refers to internal relative CVIs. In this paper, to avoid any confusion between internal and external CVIs, we call external CVIs *partition similarity measures.*

Most novel PSC techniques or application (e.g. [22, 23]) simply adopt a certain relative CVI as their fitness function without any justification for the selection, but in general any PSC technique can be easily modified to

use any CVI as its fitness function. The fitness function selection can significantly affect the clustering results and should be thus carefully considered. A poorly selected fitness function will lead to undesired partitions no matter how advanced the applied PSC technique is. As different CVIs are designed to address different data distributions, they may lead to completely different partitions. A certain CVI may be able to handle unbalanced data well, while having a moderate performance on overlapping clusters. Therefore, it would be best to select the CVI to be used as a fitness function according to the data type to be clustered. However, the data properties are usually unknown to the user of a PSC method. When this is the case, it would be recommendable to use several CVIs [14] or a combined CVI [24, 25]. Whether using a single CVI or a group of CVIs, it is important to know how these CVIs are performing on different data types.

Most CVIs work best with spherical clusters having equal number of data items, but there are mechanisms to accommodate unbalanced data, anisotropic data distribution, or arbitrarily shaped clusters, while usually such mechanisms come with a trade-off and the performance may suffer in other cases. Similar to K-means, in PSC with centroid-based particle encoding, the data space is always partitioned into Voronoi cells. Therefore, it is not meaningful to use CVIs that are able to handle data with peculiar cluster shapes or significant anisotropy. On the other hand, the ability to handle unbalanced data is usually desired.

Furthermore, when the number of clusters is unknown, this raises an important question: how the selected CVI behaves with varying number of clusters? It is important to note that not all the CVIs reach their optimal value for the optimal number of clusters, but for some CVIs different statistics such as maximum increase/decrease or maximum/minimum of the second differences, are better suited to detect the optimal number of clusters [26, 27]. However, PSO can only optimize its fitness function and, therefore, the optimum of the selected CVI should also indicate the optimal number of clusters in order to simultaneously optimize the number of clusters and obtain the corresponding optimal partition. For this reason, CVIs which are performing well in general comparisons may not be successful as PSC fitness functions.

When considering the behavior of a certain CVI for different numbers of clusters, it should be also noted that clustering is often used as a part of a more complicated system (e.g. ECG classification in [28] or training of radial basis function networks in [19]). Commonly the obtained clusters will be

afterward labeled or processed in a certain manner and, generally, it is not a problem to give the same label or treatment for several clusters. Instead, it is more problematic if the items assigned to a single cluster do not represent a single natural class. Therefore, if the partitions are otherwise meaningful, CVIs that have a tendency to suggest too many clusters (i.e. overclustering) can usually be preferred over those CVIs that are more apt to undercluster.

3.2. Fitness Evaluation with Computational Centroids

Most existing CVIs depend on the cluster centroids in one way or another. As the purpose of the PSO fitness function is to evaluate the fitness of a particle and in PSC each particle represents a set of candidate cluster centroids, it seems natural to simply replace the cluster centroids in CVIs with centroids proposed by particle positions. This approach has been used e.g. in [2, 11, 13, 29].

However, usually the centroids proposed by particle positions are different from the computational centroids of the items assigned to the corresponding clusters as illustrated in Fig. 2. We propose to apply FECC, where the fitness function values are computed using the latter. While the traditional approach better corresponds with the actual fitness of the particle positions, the final goal of the PSC process is to find a dataset partition i.e. a cluster label for each data item. Therefore, it can be argued that, if two candidate sets of centroids result in the same final data partition, their fitness should be the same. In a situation similar to that in Fig. 2, the clustering process using the traditional way to compute the fitness score would most probably later converge so that particle positions and computational centroids match, but the computational effort required would be wasted in the sense that the final data partition would not further improve. Using FECC this computational effort is saved. In real clustering problems, clusters are commonly not that well separated and moving the particle positions to match with the computational centroids would result in different cluster memberships (this is a well-known fact from the functioning principle of K-means). We believe that in such situations FECC has a greater potential to guide the PSC process toward better partitions. With the traditional approach, a promising partition may be penalized and lost only because the particle positions are somewhat outside the data distribution.

FECC can be used with any PSC method, where fitness evaluation is performed using a CVI depending on the cluster centroids, as the only modification needed is the computation of the mean of the data items assigned to

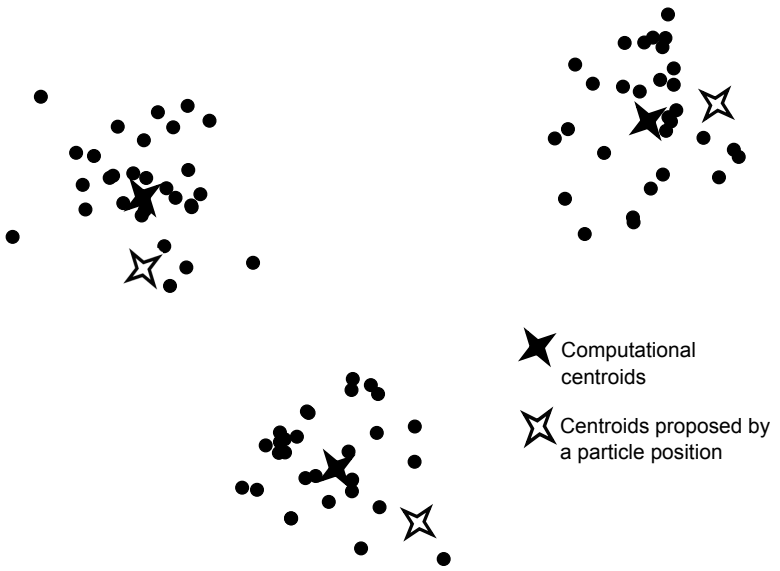


Figure 2: Centroids proposed by a particle position and computational centroids of items belonging to the corresponding clusters usually differ

a certain cluster before computing the value of the selected CVI. In some earlier PSC applications, FECC has been possibly used, as the details of fitness evaluation are often discarded, but to the best of our knowledge, it has not been specifically proposed, evaluated, or compared against the traditional fitness evaluation in any earlier publication.

4. Clustering Validity Indices

4.1. CVI Definitions

In this section, we introduce the 17 CVIs selected for comparative evaluations. All the selected indices except SIL, Dunn, and Ratkowsky & Lance directly depend on the cluster centroids and they are used in fitness evaluation in the traditional way and with the FECC approach. The considered fitness functions are listed in Table 1. The FECC version is denoted with an asterisk after the abbreviation (e.g. BH*). As PSC fitness functions, CVIs are minimized even if another approach (e.g. knee-point detection) would better indicate the optimal number of clusters. The indices to be maximized are negated. Below, the formulas and explanations for all the CVIs are given. The notations used in the CVI formulas are defined in Table 2.

A CVI commonly used with particle swarm clustering (e.g. in [1, 29]) is the **quantization error** defined as

$$QE = \frac{1}{K}\sum_{i=1}^{K}\sum_{\mathbf{x}\in C_i}\frac{\|\mathbf{x}-\mathbf{m}_i\|}{n_i}. \tag{2}$$

Lower values of the quantization error indicate better partitions.

Ball & Hall index ([30]) is simply the average distance of each item to their respective cluster centroid:

$$BH = \frac{1}{n}\sum_{i=1}^{K}\sum_{\mathbf{x}\in C_i}\|\mathbf{x}-\mathbf{m}_i\|. \tag{3}$$

Lower values of the index indicate better partitions. When this index has been previously used to decide the number of clusters, the maximum difference between two hierarchy levels has been typically used as the indicator of the best number of clusters [31, 32].

Several commonly used CVIs aim at minimizing the within-cluster sum-of-squares, ssw. Many of them also simultaneously attempt to maximize the

Table 1: PSC fitness functions compared in this article. Asterisks denote FECC approach.

Notation	CVI	Formula
$QE, QE*$	Quantization error	(2)
$BH, BH*$	Ball & Hall index	(3)
$CH, CH*$	Calinski-Harabasz index	(4)
$HA, HA*$	Hartigan index	(5)
$Xu, Xu*$	Xu index	(6)
$WB, WB*$	WB index	(7)
RL	Ratkowsky & Lance index	(8)
SIL	Silhouettes index	(9)
$I, I*$	I index	(12)
$DB, DB*$	Davies-Bouldin index	(14)
$mDB, mDB*$	Modified DB index	(17)
$\Gamma, \Gamma*$	Normalized modified Hubert Γ index	(18)
$CS, CS*$	CS index	(19)
DU	Dunn index	(20)
$DU_{23}, DU_{23}*$	Dunn variant DU_{23}	(23) and (26)
$DU_{33}, DU_{33}*$	Dunn variant DU_{33}	(24) and (26)
$DU_{53}, DU_{53}*$	Dunn variant DU_{53}	(25) and (26)

Table 2: Notations used in CVI formulas

Explanation	Notation and formula
Data item	$\mathbf{x}$
Set of all items	C
Total mean vector	$\mathbf{m} = \frac{1}{n} \sum_{\mathbf{x} \in C} \mathbf{x}$
Number of items	n
Data dimension	N
Number of clusters	K
i^{th} cluster	C_i
Number of items in cluster C_i	n_i
Centroid of cluster C_i	$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in C_i} \mathbf{x}$
Sum-of-squares for cluster C_i	$ssw_i = \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{m}_i\|^2$
Within-cluster sum-of-squares	$ssw = \sum_{i=1}^{K} ssw_i$
Between-cluster sum-of-squares	$ssb = \sum_{i=1}^{K} n_i \|\mathbf{m}_i - \mathbf{m}\|^2$
Total sum-of-squares	$sst = \sum_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{m}\|^2$

between-cluster sum-of-squares, ssb. Since the total sum-of-squares, $sst = ssw + ssb$, is constant for a given dataset, these objectives are overlapping [33]. This group of CVIs includes **Calinski-Harabasz index** or **CH index** (as given in [31])

$$CH = \frac{ssb}{ssw} \times \frac{n-K}{K-1} \tag{4}$$

Hartigan index (as given in [31])

$$HA = \log \frac{ssb}{ssw} \tag{5}$$

Xu index [34]

$$Xu = N \log \left(\sqrt{\frac{ssw}{Nn^2}} \right) + \log K \tag{6}$$

WB index [35]

$$WB = K \frac{ssw}{ssb} \tag{7}$$

and **Ratkowsky & Lance index** (as given in [31, 36])

$$RL = \frac{\frac{1}{N} \sum_{d=1}^{N} \sqrt{\frac{ssb_d}{sst_d}}}{\sqrt{K}} \tag{8}$$

where ssb_d and sst_d are separately computed for each dimension of the data. In our experiments, we removed any dimension for which $sst_d = 0$ for this index. As evident from the definitions, CH, Hartigan, and Ratkowsky & Lance indices are to be maximized, while Xu and WB indices should be minimized. In [31, 32], the optimal number of clusters is directly decided using the maximum value of CH and Ratkowsky & Lance indices, while for Hartigan index the maximum difference is used. In [26], the second differences are also used for CH and Ratkowsky & Lance indices, but in that study the definition of Ratkowsky & Lance index was given without the denominator $\sqrt{K}$ in Eq. (8). The inventors of WB index [35] claim that it should determine the optimal number of cluster directly by its minimum value, while for other ssw-based indices a knee-point is a better indication.

Silhouettes or SIL index [37] can be computed as the average of silhouette widths s_i of all data items $\mathbf{x}_i$,

$$SIL = \frac{1}{n} \sum_{i=1}^{n} s_i = \frac{1}{n} \sum_{i=1}^{n} \frac{b_i - a_i}{\max(a_i, b_i)} \tag{9}$$

where a_i is the average distance of item $\mathbf{x}_i$ to the other items belonging to the same cluster C_j,

$$a_i = \frac{1}{n_j - 1} \sum_{\substack{\mathbf{x}_k \in C_j, \\ k \neq i}} \|\mathbf{x}_i - \mathbf{x}_k\| \tag{10}$$

and b_i is the minimum average distance of item $\mathbf{x}_i$ to the items belonging to some other cluster C_h,

$$b_i = \min_{\substack{h \in \{1,\ldots,K\}, \\ h \neq j}} \frac{1}{n_h} \sum_{\mathbf{x} \in C_h} \|\mathbf{x}_i - \mathbf{x}\|. \tag{11}$$

If there is only a single item $\mathbf{x}_l$ in a cluster, s_l is set to zero. Thus, for each data item $\mathbf{x}_i$, $-1 \leq s_i \leq 1$. When s_i is close to 1, item $\mathbf{x}_i$ is well clustered, i.e. it is significantly closer to items in its own cluster than items in any other cluster. When s_i is close to -1, item $\mathbf{x}_i$ has been clearly misclustered. Thus, larger values of SIL index indicate better partitions.

I index [38] is defined as

$$I = \left(\frac{1}{K} \frac{E_1}{E_K} \max_{i,j \in \{1,\ldots,K\}} \|\mathbf{m}_i - \mathbf{m}_j\| \right)^p \tag{12}$$

where $p \geq 1$ (following [38], we set p to 2) and

$$E_K = \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{m}_i\|. \tag{13}$$

Thus, E_1 is constant for a given dataset. It is used as a "normalizing factor" to avoid extremely low index values. Partitions with larger values of I index are preferred.

Davies-Bouldin index or **DB index** [39] is one of the classical validity indices. It attempts to minimize the intra-cluster distances and maximize the inter-cluster distances for each cluster C_i as follows:

$$DB = \frac{1}{K} \sum_{i=1}^{K} \max_{j \in \{1,\ldots,K\}, j \neq i} \left(\frac{e_i + e_j}{M_{ij}} \right) \tag{14}$$

where e_i is the dispersion of cluster C_i,

$$e_i = \left(\frac{1}{n_i} \sum_{\mathbf{x} \in C_i} \|\mathbf{m}_i - \mathbf{x}\|^q \right)^{\frac{1}{q}} \tag{15}$$

and M_{ij} is the distance between the centroids of clusters C_i and C_j,

$$M_{ij} = \left(\sum_{d=1}^{N} |m_{id} - m_{jd}|^p \right)^{\frac{1}{p}}. \tag{16}$$

The minimum value of DB indicates the best partition. We set $q = 1$ and $p = 2$ similar to [31, 32, 40].

The modified DB index [41] is defined as

$$mDB = \frac{1}{K} \sum_{i=1}^{K} \frac{\max_{j=1,\ldots,K, j\neq i} (e_i + e_j)}{\min_{k=1,\ldots,K,k\neq i} (M_{ik})} \tag{17}$$

where $q = 1$ and $p = 2$ are used in Eq. (15) and Eq. (16).

The normalized modified Hubert Γ index [42] is defined as

$$\Gamma = \frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{(P_{ij} - \mu_P)(Q_{ij} - \mu_Q)}{\sigma_P \sigma_Q} \tag{18}$$

where $M = n(n-1)/2$, P is the proximity matrix of the dataset, Q is a $n \times n$ matrix, whose element Q_{ij} is the distance of the centroids of the two clusters, which include data items $\mathbf{x}_i$ and $\mathbf{x}_j$, μ_P, μ_Q, σ_P, and σ_Q are the means and variances of P and Q matrices. Higher values of the index indicate better clustering results. According to [43], knee-point detection should be applied for this index to decide the optimal number of clusters.

CS index [44] is defined as

$$CS = \frac{\sum_{i=1}^{K} \left(\frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} \max_{\mathbf{x}_l \in C_i} \|\mathbf{x}_j - \mathbf{x}_l\| \right)}{\sum_{i=1}^{K} \left(\min_{\substack{j \in \{1,\ldots,K\}, \\ j \neq i}} \|\mathbf{m}_i - \mathbf{m}_j\| \right)} \tag{19}$$

and lower values of the index indicate better partitions.

Dunn index [45] is defined as

$$DU = \frac{\min_{i,j \in \{1,\ldots,K\}, i \neq j} \delta(C_i, C_j)}{\max_{k \in \{1,\ldots,K\}} \Delta(C_k)} \tag{20}$$

where $\delta(C_i, C_j)$ is the distance of two clusters C_i and C_j defined as

$$\delta(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} \|\mathbf{x} - \mathbf{y}\| \tag{21}$$

and $\Delta(C_k)$ is the cluster diameter defined as

$$\Delta(C_k) = \max_{\mathbf{x}, \mathbf{y} \in C_k} \|\mathbf{x} - \mathbf{y}\|. \tag{22}$$

Higher values of the index indicate better partitions.

Bezdek and Pal [43] proposed five alternative ways to compute $\delta(C_i, C_j)$ and two alternative ways to compute $\Delta(C_k)$. Different combinations of the original and alternative formulas give 17 variants of Dunn index. The following alternatives for cluster distance and cluster diameter were shown to yield some of the best results:

$$\delta_2(C_i, C_j) = \max_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} \|\mathbf{x} - \mathbf{y}\| \tag{23}$$

$$\delta_3(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\mathbf{x} \in C_i} \sum_{\mathbf{y} \in C_j} \|\mathbf{x} - \mathbf{y}\| \tag{24}$$

$$\delta_5(C_i, C_j) = \frac{1}{n_i + n_j} \left(\sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{m}_j\| + \sum_{\mathbf{y} \in C_j} \|\mathbf{y} - \mathbf{m}_i\| \right) \tag{25}$$

and

$$\Delta_3(C_k) = 2 \frac{\sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \mathbf{m}_k\|}{n_k}. \tag{26}$$

In this paper, we will consider variants $\mathbf{DU_{23}}$, $\mathbf{DU_{33}}$, and $\mathbf{DU_{53}}$.

4.2. Earlier CVI Comparisons

4.2.1. Description of Comparative Evaluations

Over the years, several comparative studies over different CVIs have been published. In a classical study of Milligan and Cooper [31], 30 different CVIs were compared. They applied hierarchical clustering using only simple artificial datasets which contained at most five distinct nonoverlapping clusters. Each dataset had only 50 items and data dimensionality was set to 4, 6, or 8. Both balanced and unbalanced classes were considered. Milligan and Cooper mainly compared CVIs in terms of the numbers of clusters suggested by

them, but they also verified using the Jaccard index and the Adjusted Rand statistic that the obtained partitions exhibit desired properties. Another extensive CVI comparison by Dimitriadou et al. was published in [26]. In this study, 15 CVIs were compared over 162 12-dimensional binary datasets using K-means and Hard Competitive Learning (HCL). Each dataset had 6000 items, the optimal cluster number was 4-6 and both balanced and unbalanced classes were used. Besides the number of clusters obtained, Dimitriadou et al. compared CVIs in terms of absolute and relative profile identification as well as classification rate. Many other CVI comparisons, which use more complicated datasets including real-life datasets, just adopt the assumption that the obtained clusters exhibit desired properties if the suggested cluster number is correct without separately verifying the assumption [43, 46, 47, 48]. This approach is criticized in [32, 49].

Vendramin et al. [32] state that CVIs should not be compared using only resulting cluster numbers, because beside the (near)optimal partition with the correct cluster number, there exist naturally several suboptimal partitions with the same number of clusters. Thus, finding the correct number of clusters may not guarantee a high quality clustering. Similarly, they state that the error made by a certain CVI should not be measured in terms of the absolute difference between the suggested cluster number and the optimal cluster number, because it is possible that a partition with a wrong cluster number is still closer to the optimal partition than a partition with the correct number of clusters. Furthermore, they criticize the common approach to assess each CVI solely based on the correctness (with respect to the number of clusters) of the partition elected as the best one according to that index. They state that it is also important to assess whether the indices can meaningfully rank the suboptimal partitions. As an overall solution to these concerns, they propose that for each dataset one should generate several partitions with varying qualities and numbers of clusters, compute CVI values for each partition, and finally evaluate the correlation between the CVI values and the values obtained using a partition similarity measure to evaluate the similarity with the ground-truth data. In [32], the classical study of [31] was first reproduced with 40 different CVIs and with a different criterion to select the optimal number of clusters. The new criterion they adopted was the difference between hierarchy levels. Then the same CVIs were compared on the same datasets using the proposed comparison protocol. The results were indeed quite different. Finally, they conducted a more comprehensive comparison on 972 datasets of 500 items using the proposed approach. In

this larger comparison only 14 CVIs were considered.

Gurrutxaga et al. [49] share the concerns mentioned above over the methodology commonly used in CVI comparisons, but they formulate their concerns somewhat differently. As the main problem, they see "the algorithm correctness assumption", i.e., the assumption that, if an algorithm can produce partitions with different cluster numbers, it will produce them so that the partition with the correct cluster number fits the data better than any other partition. If this is not true and a partition with a wrong cluster number is closer to the optimal partition, it is not fair to assume that CVIs should be able to still predict the correct number of clusters. Therefore, Gurrutxaga et al. suggest that instead of comparing CVIs in terms of the cluster number, one should compare the similarity of the correct partition and the partition receiving the highest CVI value using a partition similarity measure. To get more robust results, they suggest combining the results obtained by several similarity measures, several clustering algorithms, and using several different datasets with varying properties. In [49], they compare seven well-known CVIs over seven synthetic datasets and three datasets from real applications using both traditional and proposed methodologies. The results show that the obtained CVI ranking is indeed different with different methodologies. In [40], the proposed methodology was applied to compare 30 CVIs over 720 synthetic and 20 real datasets.

While we agree with the above principles, our objectives are naturally somewhat different than those discussed in [32, 49]. Both works attempted to find out, which CVI can select the best partition among several possible partitions or rank these partitions, while we want to know which CVI can be most successfully used for fitness evaluation in PSC. Nevertheless, the above listed reasons explain, why the CVI ranks obtained in earlier comparisons are varying significantly and why the majority of the results may not really describe which CVIs produce best partitions. Furthermore, it can be concluded that the best ranking CVIs in general comparisons may not be the ones most suitable for fitness evaluation in PSC. Thus, the most relevant CVI comparison for this study has been published in [14]. In this work, the authors compared 8 common CVIs over 6 synthetic and 4 real datasets using DEPSO for dynamic PSC. Beside the cluster number, they used four different partition similarity measures to evaluate the quality of the obtained partitions.

4.2.2. Overview of Earlier Comparison Results

An overview of the results obtained in the earlier comparisons of the CVIs introduced in Section 4.1 is shown in Table 3, where lighter color means a better relative performance in the comparison and 'X' means that the CVI was not considered at all in the comparison. As explained above, the applied methods and evaluation criteria vary a lot and, therefore, a clear comparative evaluation of the results from different comparisons is not meaningful. Also it should be noted that the ranks shown in the table are somewhat arbitrary. The rank may be based on only one of several criteria and in some cases the differences between the CVIs were insignificantly small, but we still used them to rank the CVIs.

Quantization error has not been considered in the earlier comparisons. **Ball & Hall index** was not successful in the classical study of Milligan and Cooper [31]. However, when the study was reproduced in [32] with a different criterion to decide the optimal hierarchy level this CVI became one of the best in the comparison (6a). With the new comparison approach proposed by Vendramin et al., this CVI was less successful (6b). In [26, 35], a different formula was used for Ball & Hall index ($\frac{ssw}{K}$).

Among CVIs aiming at minimizing ssw, **CH index** was the winner of the classical comparison study of [31] and it has been successful also in the later comparisons. Interestingly, the index has produced good results when used either with the maximum [31, 32] or the maximum difference [26, 35] as the indicator of the optimal number of clusters. **Hartigan index** has not been among the top CVIs in any of the previous comparisons. **Ratkowsky & Lance** and **Xu indices** were the top two overall performers in the comparison of [26], but in that study the definition of Ratkowsky & Lance index was given without the denominator $\sqrt{K}$ in Eq. (8). Ratkowsky & Lance index was also ranked high in [32], when the proposed comparison approach was used (6b), but further analysis carried out by the authors showed that the success was misleading and the index was dominated by the number of clusters. In [31], Ratkowsky & Lance was ranked below average. **WB index** has not been considered in extensive comparisons, but in the original paper [35] it outperformed several other indices, when the ability to find the correct number of clusters was evaluated.

SIL index has been the best performing CVI in several comparisons [14, 32, 40], but it is also among the computationally most complex indices [32]. Also **I index** has been relatively successful in recent comparisons [14, 32]. In

Table 3: An overview of results obtained in earlier CVI comparisons. Lighter color means better relative success.

	QE	BH	CH	HA	Xu	WB	RL	SIL	I	DB	mDB	Γ	CS	DU	DU23	DU33	DU53
1	X				X	X		X	X		X	X	X	X	X	X	X
2	X					X		X	X		X	X	X	X	X	X	X
3	X	X			X	X	X		X	X	X	X	X	X	X	X	X
4	X	X	X	X	X	X	X	X	X		X		X				
5	X	X		X	X	X	X	X			X	X	X		X	X	X
6a	X				X	X					X	X	X				
6b	X				X	X					X	X	X		X	X	X
6c	X	X		X	X	X					X	X	X			X	X
7	X	X		X	X	X	X		X			X			X		
8	X	X		X	X	X	X	X	X		X	X	X		X	X	X
9	X	X		X	X	X	X				X	X			X		
10	X						X		X		X	X			X	X	X

[1] [31]

[2] [26], ranking based on Table 2

[3] [47], ranking according to the overall number of runs where the true cluster number was found on the first 4 datasets

[4] [43], ranking based on Table VIII

[5] [48], ranking based on Table 2

[6] [32] a) ranking based on the reproduced study of Milligan and Cooper, Table 9, b) ranking based on the same comparison with the proposed approach, Table 10, c) ranking based on the more comprehensive comparison with the proposed approach, Table 17

[7] [40], ranking based on Table 2

[8] [49], ranking based on Table 4

[9] [14], ranking based on average rank in Table IV

[10] [35], ranking based on Table 2

[31, 32, 40], **DB index** was used setting $q = 1$ and $p = 2$, while in [26], e_i was replaced by ssw_i. In all these comparisons, DB index was among the upper mid-level performers. In the extensive comparison of [40], also a modified version introduced in [41] was evaluated and found to be performing better than the original version. **The normalized modified Hubert Γ index** has been considered only in [43], where it was an average performer. **CS index** has been evaluated in [14] and [40] to have worse than average performance.

In most earlier comparisons, **Dunn index** has not been among the best performing CVIs (e.g. [40, 32]), whereas in [14, 49] it was relatively successful. In [43], [32] (6a), and [40], the considered **Dunn variants** clearly outperformed the original Dunn index. A contrary result was obtained in [14] were the original Dunn index was better than DU_{33} and DU_{53} variants.

5. Experimental Results

5.1. Partition Similarity Measures

We use three different partition similarity measures (external CVIs) to evaluate the similarity of the ground-truth partitions and the obtained partitions: the Jaccard index [50], the Adjusted Rand index [51], and the labeling error. If $\mathbf{P}_{real}$ is the ground-truth partition and $\mathbf{P}_{PSO}$ is the partition obtained by PSC, for each pair of items $\mathbf{x}_i, \mathbf{x}_j, i \neq j$ there are four possible cases: 1) they belong to the same cluster in both $\mathbf{P}_{real}$ and $\mathbf{P}_{PSO}$, 2) they belong to the same cluster in $\mathbf{P}_{real}$, but different clusters in $\mathbf{P}_{PSO}$, 3) they belong to different clusters in $\mathbf{P}_{real}$, but to the same cluster in $\mathbf{P}_{PSO}$, or 4) they belong to different clusters in both $\mathbf{P}_{real}$ and $\mathbf{P}_{PSO}$. If a, b, c and d are used to denote the number of item pairs for each case, respectively, the Jaccard index can be defined as

$$Jaccard = a/(a + b + c) \tag{27}$$

and the Adjusted Rand index as

$$AR = \frac{M(a + d) - ((a + b)(a + c) + (c + d)(b + d))}{M^2 - ((a + b)(a + c) + (c + d)(b + d))} \tag{28}$$

where $M = a+b+c+d$. The Adjusted Rand index is corrected for randomness so that its values around zero indicate that the similarity of two partitions could be due to chance. For (near) perfectly agreeing partitions, the Adjusted

Rand index value is (close to) one. The value can also be less than zero, indicating less than chance agreement between partitions.

To compute the labeling error, we label each cluster in $\mathbf{P}_{PSO}$ according to the majority $\mathbf{P}_{real}$ label present and, using these cluster labels, we compute the percentage of mislabeled data items. We do not care about the number of clusters when computing the clustering error, i.e., in an extreme case one could assign each data item into a different cluster and still obtain a zero percent labeling error. However, the defined dimensionality range $\{d_{\min}, \ldots, d_{\max}\}$ limits the maximum number of clusters to be $d_{\max}$ and subjectively we feel that for many real life applications it does not matter if the obtained cluster number, K_{PSO}, is higher than the optimal, K_{real}, as long as $K_{PSO} << n$ and the labeling error is low.

5.2. Datasets

We created 720 synthetic datasets using the methodology introduced in [40]. All the datasets are sampled inside a fixed hypercubic window defined by coordinates $\{0, 0, \ldots, 0\}$ and $\{50, 50, \ldots, 50\}$. Similarly a reduced sampling window defined by coordinates $\{3, 3, \ldots, 3\}$ and $\{47, 47, \ldots, 47\}$ is used to sample the cluster centroids. Once a cluster centroid, $\mathbf{m}_i$ has been chosen, all cluster items are drawn from a multivariate normal distribution with mean $\mathbf{m}_i$ and identity covariance matrix. Any item outside the sampling window is resampled.

The datasets cover all the possible combinations of the parameters given in Table 4. The meaning of the number of clusters, K, and the dimensionality, d, is obvious. If the density, *den*, is set to 1, all the clusters have $n_{\min}$ items. If the density is set to 4, the first cluster has $4 * n_{\min}$ items, while all the other clusters have $n_{\min}$ items. This produces a density asymmetry, because different numbers of items are located in the approximately same volume. When the overlap, *ov*, is bounded, all the cluster centroids are drawn from a uniform distribution inside the reduced sampling window. If the strict overlap is used, the first centroid is similarly drawn from a uniform distribution inside the reduced sampling window, while the remaining centroids are set to random points located at distance $2 * ov$ from a randomly selected previously created cluster centroid, $\mathbf{m}_k$. For both overlap types, all the cluster centroids must be separated by the minimum Euclidean distance of $2 * ov$, i.e., if $\exists i \neq j, \|\mathbf{m}_i - \mathbf{m}_j\| < 2 * ov$, or the newly created centroid, $\mathbf{m}_j$, will be resampled. If a noise level, *nl*, greater than zero is used, $nl * N'$ noise items are randomly drawn from a uniform distribution inside the sampling window, where N' is

Table 4: Parameters of the created synthetic datasets. All possible combinations were used.

Parameter	Values
Number of clusters, K	2,4,8
Dimensionality, d	5,10,20
Density *den*	1,4
Minimum number of items per cluster, $n_{\min}$	100
Overlap, *ov*	2 (strict), 5 (bounded)
Noise level, *nl*	0, 0.1

the number of non-noise items in the dataset ($N' = n_{\min} * (den + K - 1)$). Using each parameter combination we created 10 datasets, resulting in a total of 720 synthetic datasets. For further details regarding the synthetic dataset creation process, the reader is referred to [40].

Similar to [40], we also evaluate fitness functions on 20 real datasets obtained from UCI Machine Learning Repository [52]. The characteristics of the datasets used in this paper are shown in Table 5. All the dataset values were normalized between 0 and 1. As the use of benchmark classification datasets to evaluate clustering algorithms may be misleading [53], we concentrate mainly on thorough analysis of the fitness function behavior on different synthetic datasets and only briefly examine whether the results are similar on the selected real datasets.

5.3. Experimental Setup

The parameters for the MD-PSO and MEPSO processes were set as follows: The swarm size, S, was set to 100 and the number of iterations to 200. The dimensionality range $\{d_{\min}, \ldots, d_{\max}\}$ was set to $\{2, \ldots, 15\}$ and the dimensional velocity range $\{dv_{\min}, \ldots, dv_{\max}\}$, to $\{-4, \ldots, 4\}$ for all the experiments. The positional search space range $(\mathbf{x}_{\min}, \mathbf{x}_{\max})$ was set according to the data values so that each element of potential cluster centroids, $\mathbf{m}_{p,j}$, defined by particle position $\mathbf{x}_p$ can vary withing the data range $[x_{\min}, x_{\max}]$ (i.e., $[0, 50]$ for synthetic datasets and $[0, 1]$ for real datasets). The positional velocity range, $(\mathbf{v}_{min}, \mathbf{v}_{max})$ was always set similarly with respect to $[-x_{\max}/10, x_{max}/10]$. The inertia weight $w(t)$ is linearly decreased from 0.9 to 0.4 during iterations of a MD-PSO run. The acceleration constants c_1 and c_2 in are set to 2 as originally suggested in [3, 54] and used widely thereafter [55].

Table 5: Characteristics of the real datasets obtained from UCI repository

Dataset	No. of objects	Dimensionality	No. of classes
Banknote auth.	1372	4	2
Breast tissue	106	9	6
Breast Wisconsin	569	30	2
Dermatology	366	33	6
Ecoli	336	7	8
Glass	214	9	6
Heart Cleveland	297	13	5
Ionosphere	351	34	2
Iris	150	4	3
Movement libras	360	90	15
Parkinsons	195	22	2
Pima Ind. Diabetes	768	8	2
Seeds	210	7	3
Segmentation	2310	19	7
Vehicle	846	18	4
Vertebral column	310	6	3
Vowel context	990	10	11
Wine	178	13	3
Winequality red	1599	11	6
Yeast	1484	8	10

For each examined fitness function, we run both MEPSO and MD-PSO 10 times for each synthetic dataset and 100 times for each real dataset. Then we compute the average performance scores using each partition similarity measure for each evaluated fitness function on each dataset. For each dataset, we rank the fitness functions based on their performance and finally we evaluate the overall results using the average ranks as recommended in [56].

5.4. Results of Comparative Evaluations

The overall results for synthetic datasets are given in Fig. 3. The top five fitness functions in our comparison are all using the proposed FECC approach, along with Xu, WB, Dunn variant DU_{23}, Ball & Hall, and CH indices. Also for the majority of the other CVIs, which can be applied using FECC, the FECC approach has produced better results than the traditional way to carry out fitness evaluation. This is a significant outcome suggesting that the previously applied PSC algorithms can generally be improved by simply modifying the algorithms to use FECC.

The top two fitness functions in Fig. 3 are interesting. Xu index was ranked second by [26], but it has not been considered in more recent comparisons. WB index is a relatively new index and it has not been previously considered in any major CVI comparisons. For the other top performing fitness functions there are more comparison results available. Ball & Hall index and Dunn variant DU_{23} were successful also in the comparison conducted in [32] and CH index has been consistently successful in many comparisons.

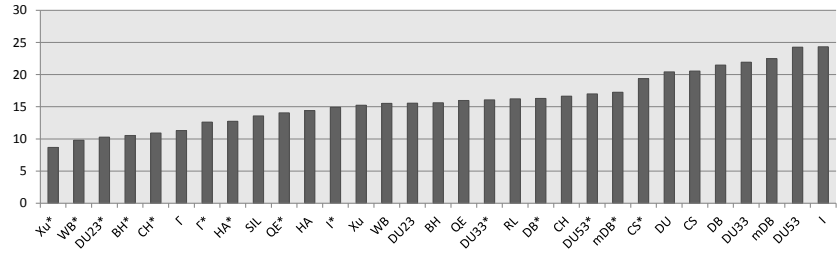


Figure 3: Average ranks over all synthetic datasets, all partition similarity measures, and both PSC algorithms

In Fig. 4, we show the average ranks based on a single partition similarity measure at a time. It can be seen that for most fitness functions almost the

same ranks have been obtained using different partition similarity measures, but some fitness functions are ranked clearly higher when the labeling error is used as the evaluation method. Such fitness functions include Ball & Hall, Hartigan, quantization error, and Γ index with FECC and at some level Ball & Hall, Hartigan, and quantization error applied the traditional way. As explained above, a better ranking obtained using the labeling error probably means that the CVIs tend to overcluster. While the other used partition similarity measures penalize overclustering, the labeling error can still find the partition successful, if the clusters are uniform.

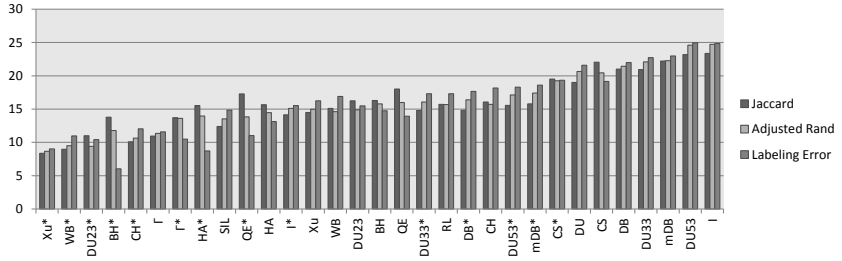


Figure 4: Average ranks over all synthetic datasets and both PSC algorithms obtained using different partition similarity measures for evaluation

In Fig. 5, we show the average ranks separately for the MEPSO and MD-PSO algorithms. As the main difference between the two algorithms is in their way to select the optimal number of clusters, we show also the average errors on the number of clusters found in Fig. 6 for the MEPSO algorithm and Fig. 7 for the MD-PSO algorithm. In Fig. 8, we show the number of times each fitness function proposed the correct number of clusters with each PSC algorithm. Figures 6-7 do confirm that the particle encoding used with MEPSO often leads to underclustering, while for MD-PSO the average cluster number is clearly higher. The figures also show that FECC almost always (the only exception being CS index for MEPSO) results in higher cluster numbers than the traditional way of fitness evaluation. Thus, it is not surprising that, according to Fig. 5, FECC is relatively more successful with MEPSO. The difference is clearest for Ball & Hall, Hartigan, quantization error, and Γ index, which were already found to have a tendency to overcluster.

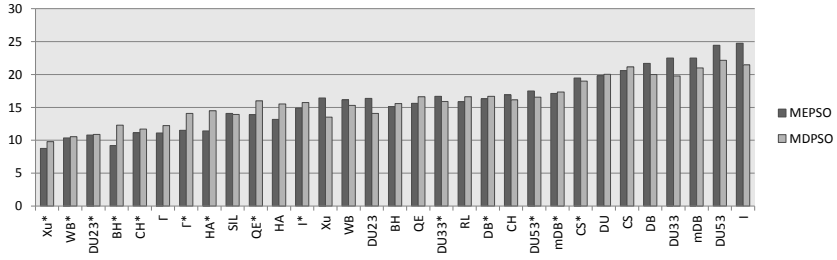


Figure 5: Average ranks over all synthetic datasets separately for both PSC algorithms

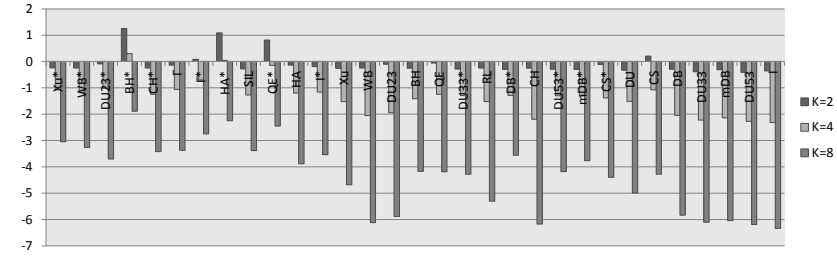


Figure 6: Average error on the number of clusters found when using the MEPSO algorithm

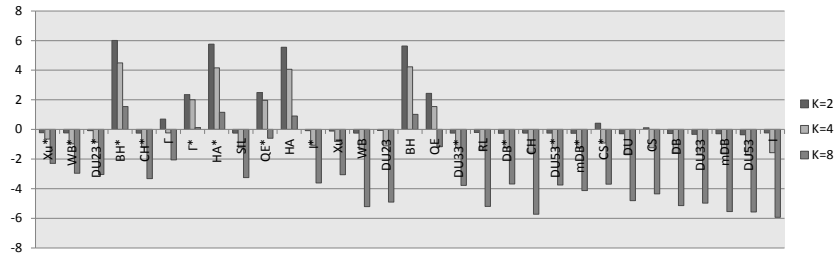


Figure 7: Average error on the number of clusters found when using the MD-PSO algorithm

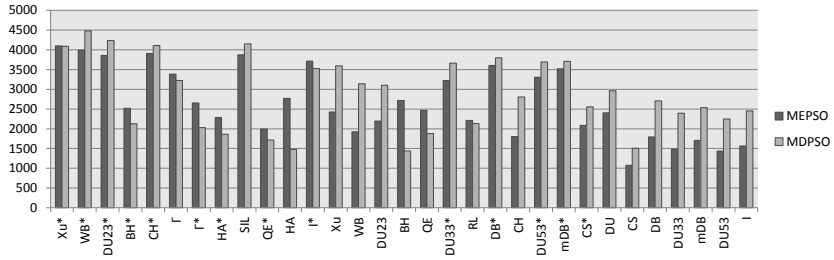


Figure 8: Number of runs out of 7200, when the correct number of clusters was found with each PSC algorithm

Fig. 8 clearly shows that evaluating fitness functions based on their ability to find the correct number of clusters would result in quite a different ranking. For example, SIL index and I and Davies-Bouldin indices with FECC would be ranked clearly higher, while Ball & Hall index would be among the poorest performing indices. Most fitness functions produce slightly more often the correct number of clusters with the MD-PSO algorithm, but the opposite is true for those fitness functions resulting in overclustering in Fig. 7.

For Ball & Hall, Hartigan, and Γ indices, the previous works have recommended using the maximum difference to find the correct cluster number, which is not followed when they are used as PSC fitness functions. Indeed, the number of correctly proposed cluster numbers is low for all of these. Nevertheless, these indices are ranked higher than several indices with a better ability to detect the correct cluster numbers. This is true even when evaluated by the Jaccard index or the Adjusted Rand index, which penalize overclustering. Thus, their ability to locate proper cluster centroid positions must be competitive and they are good fitness function choices especially when overclustering is not a problem. Also CH and Ratkowsky & Lance indices were used with the maximum difference in [26]. Ratkowsky & Lance is among the poorest cluster number detectors, but especially the FECC version of CH index is one of the top indices, when evaluated according to its ability to find correct cluster numbers.

In figures 9-13, the average ranks are shown individually for each dataset property listed in Table 4. All these results are computed over all the partition similarity measures and both PSC algorithms. It should be kept in mind here that even if a fitness function yields a better ranking with more difficult

data properties (e.g. with overlap or noise) it does not mean that its absolute performance is better in those cases. The performance of all fitness functions suffers from noise or overlapping clusters, but some suffer relatively less than others and, therefore, their average ranks get higher in these cases. Fig. 9 shows that the most of the top 5 fitness functions (Xu, WB, Dunn variant DU_{23}, and CH indices with FECC) are consistently top performing for all numbers of clusters. Both versions of Ball & Hall, Hartigan, and quantization error and Γ index with FECC have higher ranks when the number of clusters increases. This is not surprising as for $K = 8$ they undercluster less likely than other fitness functions as shown in Fig. 7. Data dimensionality does not have a significant influence on the fitness function ranking as shown in Fig. 10. Data density affects the fitness function ranking even less (Fig. 11). Overlap and noise have a larger impact on the fitness function ranking. Fig. 12 shows that SIL index and I, Davies-Bouldin, Dunn variant DU_{53}, and the modified Davies-Bouldin with FECC suffer relatively the most from the overlapping clusters, whereas especially Dunn variant DU_{23} with both approaches, Ball & Hall with FECC, and WB and CH indices with the traditional approach have higher ranks when clusters are overlapping. Similarly, Fig. 13 shows that one or both versions of Xu, WB, CH, and the normalized modified Hubert Γ indices lose ranks when noise is introduced, while Ball & Hall, Dunn variant DU_{23}, and Hartigan with FECC and quantization error with both approaches are ranked higher. Interestingly the simplest index, Ball & Hall performs always relatively better when the clustering task gets more difficult in some way.

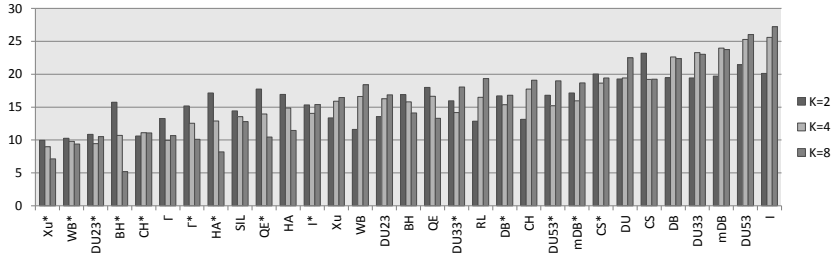


Figure 9: Average ranks over synthetic datasets with different cluster numbers

Finally, figures 14-16 show the average ranks on the real datasets. Now WB index with FECC has the overall best performance. The second place

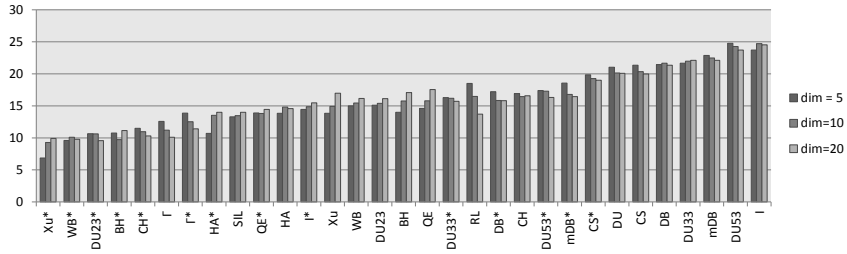


Figure 10: Average ranks over synthetic datasets with different data dimensions

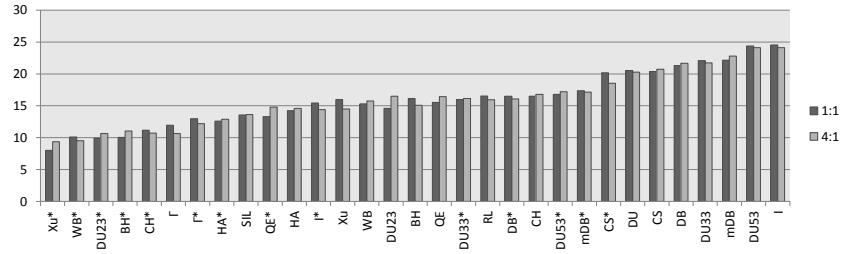


Figure 11: Average ranks over synthetic datasets with different cluster densities

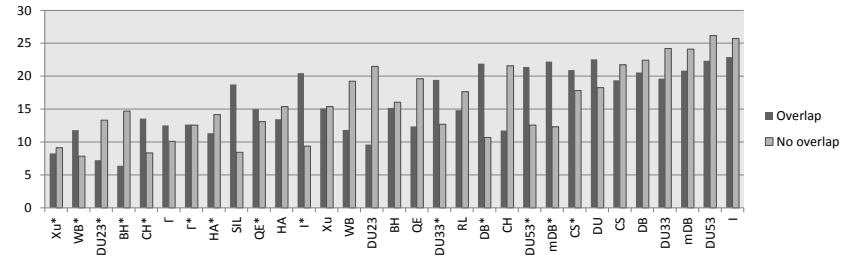


Figure 12: Average ranks over synthetic datasets with and without overlap

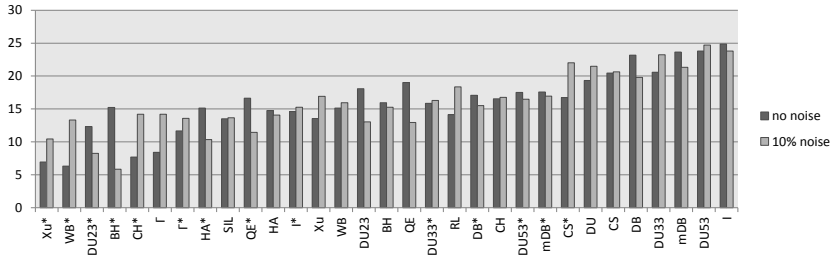


Figure 13: Average ranks over synthetic datasets with and without noise

is occupied by Ratkowsky & Lance index, which was not among the top performers for synthetic datasets. These results must be interpreted with caution and with some reserves. For example, it is possible that a class consists of several natural clusters (groupings in the feature domain) and, therefore, the best clustering result may not coincide with the best classification result. From Fig. 15 we conclude that such a scenario likely occurs at least for some datasets. In that case the labeling error could be low for indices that tend to overcluster, if they correctly detect the natural clusters, but the scores of the other partition similarity measures would suffer significantly. Fig. 15 shows that for certain fitness functions (e.g. Xu*, BH*, BH, HA*, HA) the rank based on the labeling error is clearly better than the ranks based on the Adjusted Rand or Jaccard indices. Fig. 16 shows that there are no major differences between the two PSC algorithms.

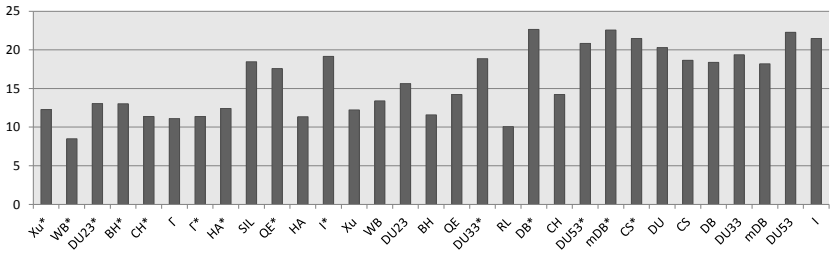


Figure 14: Average ranks over all real datasets, all partition similarity measures, and both PSC algorithms

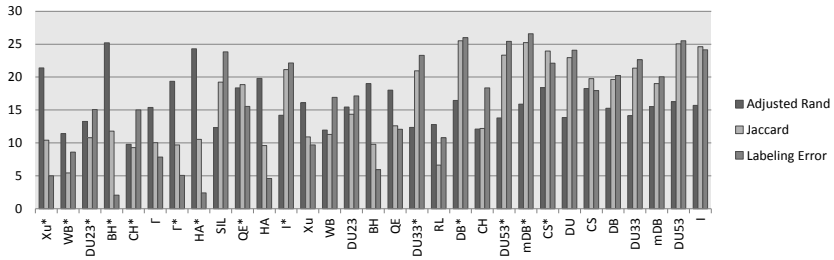


Figure 15: Average ranks over all real datasets and both PSC algorithms obtained by using different partition similarity measures for evaluation

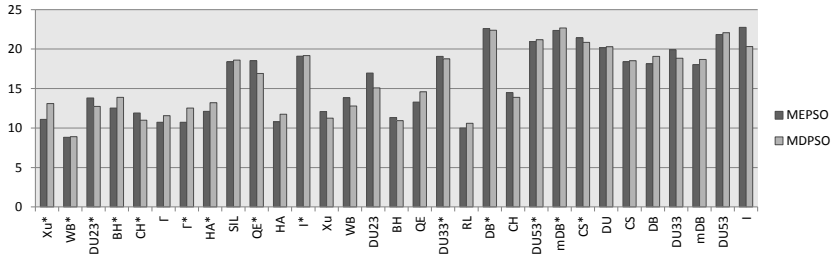


Figure 16: Average ranks over all real datasets separately for both PSC algorithms

5.5. Statistical Evaluation of Results

We first evaluated whether the differences between the compared fitness functions are statistically significant overall. We used the Friedman test as recommended in [56]. The results confirm that the differences are statistically significant when all the datasets are considered. The differences are statistically significant without exceptions also when the results are separately evaluated for each subset of datasets selected so that they share a data property (i.e., similar to separate cases handled in figures 9-13).

Next, we conducted a pairwise comparison between each pair of fitness functions by applying Wilcoxon Signed-Ranks test [56]. We evaluated the statistical significance of the observed differences separately for each partition similarity measure and for both PSC techniques. In Table 6, we show the cases where the difference between the two fitness functions is statistically significant according to at least 2 partition similarity measures. A negative number denotes that the lower ranked fitness function is statistically better in that case. The results for each pair of fitness functions using the same CVI with the traditional and FECC approaches are bolded. In the upper triangle, the results are given for MD-PSO and in the lower triangle for MEPSO.

In the most cases, the differences between the fitness function pairs are statistically significant according to all the applied partition similarity measures. The above discussed differences between the two PSC algorithms are also statistically visible. For MD-PSO algorithm alone, the ranks would be worse, e.g., for Ball & Hall, Hartigan, and quantization error applied using FECC, which is shown by negative values against some other fitness functions. Similarly, for MEPSO the ranks would be worse, e.g., for WB, Dunn variant DU_{23}, and CH applied using the traditional approach. There are also significant differences between the considered partition similarity measures and sometimes opposite statistically significant difference is detected with different measures. The labeling error, for example, does not penalize overclustering but penalizes underclustering more than the other measures, which may lead to opposite ranking of under/overclustering fitness functions.

The bolded significance evaluation results between the fitness function pairs applied using both the traditional and the FECC approaches over the same CVI show that in the most cases (Xu, WB, CH, I, Davies-Bouldin, modified Davies-Bouldin, CS, Dunn variants DU_{23}, DU_{33}, and DU_{53}) the FECC approach is better for both PSC algorithms with a statistical significance according to all the three partition similarity measures. For MEPSO, this is true also for Ball & Hall and quantization error. In the remaining

Table 6: Number of partition similarity measures having a statistically significant difference between the two fitness functions according to Wilcoxon Signed-Ranks test with a 0.05 significance level. The number is shown only when two or three measures agree on the significance. Negative numbers mean that lower ranked fitness function is better. In the upper/lower triangle, the numbers for MD-PSO/MEPSO are given.

	Xu*	WB*	DU23*	BH*	CH*	Γ	Γ*	HA*	SIL	QE*	HA	I*	Xu	WB	DU23	BH	QE	DU33*	RL	DB*	CH	DU53*	mDB*	CS*	DU	CS	DB	DU33	mDB	DU53	I
Xu*	-	2	2	2	3	3	3	2	3	3	3	3	**3**	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
WB*	3	-	3	2	3	2	2	2	3	3	2	3	3	**3**	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
DU23*	3	3	-	2	2	-2	2	2	3	3	2	3	3	3	**3**	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
BH*	2	-	-	-	-2	-2	-2	3	-	2	-	-	-2	-	2	**-**	2	-	-	2	-	2	2	2	2	2	2	2	2	3	2
CH*	3	3	-	-	-	-	2	2	3	3	2	3	3	3	3	2	3	3	3	3	**3**	3	3	3	3	3	3	3	3	3	3
Γ	3	2	-2	-	-	-	**2**	2	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Γ∗	2	2	-2	-	-	**-**	-	2	-	2	3	2	-2	2	2	3	3	2	3	2	3	2	2	3	3	3	3	3	3	3	3
HA*	2	2	2	3	2	2	2	-	-2	-	**-2**	-2	-2	-2	-2	-2	-	-2	-2	-	-2	-2	-	2	2	2	2	2	2	2	2
SIL	3	3	2	2	3	3	3	-	-	2	2	3	-2	2	-	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3
QE*	3	3	3	3	3	3	3	2	-	-	-	-	-3	-2	-2	-	**2**	-2	-	2	-	-	2	3	3	3	3	3	3	3	3
HA	3	3	3	2	3	3	3	**-**	-	-2	-	-	-2	-	-	-2	2	-	-	2	-	2	2	2	2	2	2	3	3	3	3
I*	3	3	3	2	3	3	3	2	3	2	2	-	-3	-	-2	-2	-	-	2	3	2	2	3	3	3	3	3	3	3	3	**3**
Xu	**3**	3	3	3	3	3	3	3	3	3	3	2	-	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
WB	3	**3**	3	3	3	3	3	3	3	3	3	3	3	-	-	2	2	-	-	-	3	-	3	3	3	3	3	3	3	3	3
DU23	3	3	**3**	3	3	3	3	3	3	3	3	3	3	2	-	-	2	-	-	2	3	2	2	3	3	3	3	3	3	3	3
BH	3	3	3	**3**	3	3	3	2	3	2	3	2	-3	-3	-3	-	2	-	-	2	-	2	2	3	3	3	3	3	3	3	3
QE	3	3	3	3	3	3	3	3	3	3	**3**	3	-	2	-2	-3	3	-	-2	-	-	-	-	2	3	3	3	3	3	3	3
DU33*	3	3	3	3	3	3	3	2	3	2	3	3	-	-2	-3	3	-	-	2	3	2	3	3	3	3	3	3	**3**	3	3	3
RL	3	3	3	3	3	3	3	3	3	-	3	2	-	-3	-3	3	2	2	-	-2	3	-2	-	3	3	3	3	3	3	3	3
DB*	3	3	3	2	3	3	3	2	3	2	3	3	-2	-2	-3	-	-	-3	2	-	2	-	3	3	3	3	**3**	3	3	3	3
CH	3	3	3	3	**3**	3	3	3	3	3	3	3	3	3	-2	3	3	3	3	3	-	-2	-	3	3	3	3	3	3	3	3
DU53*	3	3	3	3	3	3	3	2	3	2	3	3	-	-2	-3	3	2	-	-2	3	-3	-	3	3	3	3	3	3	3	**3**	3
mDB*	3	3	3	3	3	3	3	2	3	2	3	3	-2	-2	-3	2	-	-	-2	3	-3	-3	-	3	3	3	3	3	**3**	3	3
CS*	3	3	3	3	3	3	3	3	3	3	3	3	3	-	-	3	3	3	3	3	-2	3	3	-	3	**3**	-	-	3	3	3
DU	3	3	3	3	3	3	3	3	3	3	3	3	3	-	-	3	3	3	3	3	-2	3	3	-	-	-	-3	-2	-	3	3
CS	3	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	3	3	3	2	3	3	**3**	2	-	-3	-2	-2	2	-
DB	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	**3**	3	3	3	3	3	2	-	-	3	3	3
DU33	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	**3**	3	3	3	3	3	3	3	3	3	-	3	3	3
mDB	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	**3**	3	3	3	3	-	-	3	3
DU53	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	**3**	3	3	3	3	3	3	3	-	-
I	3	3	3	3	3	3	3	3	3	3	3	**3**	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	-

6 cases, the partition similarity measures are not unanimous. There is no such case, where the traditional approach would be evaluated better with a statistical significance by all the measures.

We further analyzed the difference of the traditional and proposed FECC approaches by applying Wilcoxon Signed-Ranks test separately for each test case from figures 9-13. There are 12 different cases and in each case 3 different partition similarity measures are used as explained. Thus, the total number of analyzed comparisons between each fitness function pair is 36 per PSC algorithm. Table 7 shows how many times either approach was evaluated to perform better with a statistical significance for the MEPSO algorithm and Table 8 for the MD-PSO algorithm. The results support the above conclusions. For most fitness function pairs FECC is significantly better in almost all the cases and for both PSC algorithms. For Ball & Hall, the normalized modified Hubert $\mathbf{\Gamma}$, Hartigan, and quantization error the difference is not so clear. Further analysis shows that in all the cases where opposite significances are detected, the labeling error generally favors FECC, while the other partition similarity measures more often favor the traditional approach. FECC is more successful also in the most difficult situations. When there is noise present, all the partition similarity measures always find the FECC approach better with a statistical significance. The same is often true also for overlapping clusters and larger numbers of clusters. When the clustering task is more difficult, there are more local optima and it is more probable to get stuck in a local optimum in lower dimensions. FECC seems to be beneficial is such situations.

Table 7: Number of test cases where the traditional fitness evaluation or FECC performs statistically better according to Wilcoxon Signed-Ranks test with a 0.05 significance level for the MEPSO algorithm. The total number of cases is 36.

	Xu	WB	BH	DU23	CH	Γ	HA
Traditional approach better	0	0	6	0	0	7	7
FECC approach better	36	36	27	36	33	18	18
	I	DB	DU33	mDB	DU53	QE	CS
Traditional approach better	0	2	0	2	0	5	2
FECC approach better	36	33	33	33	35	23	28

Table 8: Number of test cases where the traditional fitness evaluation or FECC performs statistically better according to Wilcoxon Signed-Ranks test with a 0.05 significance level for the MD-PSO algorithm. The total number of cases is 36.

	Xu	WB	BH	DU23	CH	Γ	HA
Traditional approach better	0	0	13	0	0	23	19
FECC approach better	33	36	18	36	36	13	15
	I	DB	DU33	mDB	DU53	QE	CS
Traditional approach better	0	2	0	3	0	7	1
FECC approach better	36	32	34	33	36	17	33

6. Conclusions

Traditionally in dynamic PSC with centroid-based particle encoding, the fitness of particle positions is evaluated using a CVI as a fitness function. Most CVIs somehow depend on the cluster centroids. In fitness evaluation, the cluster centroids are traditionally replaced by centroids proposed by a particle position. In this paper, we propose a new way to conduct fitness evaluation in PSC. In the proposed FECC approach, the actual centroids of the items belonging to the corresponding clusters are first computed and then these computational centroids are used in the fitness evaluation.

We conducted an extensive comparative evaluation of FECC against the traditional approach using 14 different CVIs as the fitness function for two different dynamic PSC algorithms, namely MEPSO and MD-PSO along with FGBF. The selected algorithms represent commonly used particle encoding schemes and they have been successfully used in PSC applications. The FECC approach was observed to bring a clear improvement to the clustering result with the most considered fitness functions. For a few fitness functions, there was no clear difference between the two approaches, but FECC was never clearly disadvantageous. Further analysis showed that FECC is even more beneficial with more difficult datasets (noise, overlapping clusters, higher cluster number). With difficult datasets there are more local optima and it is more probable to get stuck in a local optimum in lower dimensions. FECC seems to help avoid those local optima and find more optimal solutions with higher cluster numbers. Indeed, FECC was observed to produce higher cluster numbers than the traditional approach with the same CVI. As underclustering is a common problem in PSC, this explains the supremacy of FECC. The only fitness functions which did not clearly benefit from FECC were those having a tendency to overcluster already using

the traditional fitness evaluations. MEPSO benefited from FECC slightly more than MD-PSO, because with the particle encoding used in MEPSO underclustering is a more severe problem.

Our second objective was to carry out an extensive comparison between different fitness functions for PSC. The selection of a proper fitness function is critical for PSC performance, but, nevertheless, the topic has not received much attention in the PSC literature. There are general CVI comparisons, but due to reasons discussed in the paper, they cannot be assumed to directly indicate best fitness functions for PSC. We conducted an extensive comparison of 31 different fitness functions based on 17 different CVIs (fitness functions based on 14 CVIs were used using the traditional fitness evaluation and FECC). We performed our tests on 720 synthetic and 20 real datasets. The clustering performance was measured against the ground-truth partitions using three different partition similarity measures. Since averages over different dataset can be meaningless, the CVI performances over different datasets were ranked and the final comparison was conducted over the average ranks. The statistical significance of the observed differences was finally evaluated using the Friedman test for the overall evaluation and Wilcoxon Signed-Ranks test to evaluate the significance of pairwise differences.

The top two fitness functions in this comparison were Xu and WB indices with the FECC approach. Both "winner" CVIs have been excluded from other recent major comparisons. These fitness functions were followed by Dunn variant DU_{23}, Ball & Hall, and CH indices with FECC. These CVIs have been among the top performers also in other recent comparisons. However, further analysis of results shows that the ranking is not unambiguous, but there are differences between the different partition similarity measures, different data distributions, and different PSC methods. Most of these differences are related to tendency of the fitness functions to under/overcluster. The labeling error, which was one the applied measures, does not penalize overclustering and, therefore, it ranks higher the fitness functions having this tendency (e.g., Ball & Hall, Hartigan, and quantization error with FECC). If the PSC application is used only as a pre-step, for example, for classification, overclustering may be even a desired property if the resulting clusters are more uniform. On the other hand, for applications where it is important to detect the correct cluster number, completely different fitness functions should be favored. The best accuracy in detecting the correct cluster number was achieved by Xu, WB, Dunn variant DU_{23}, and CH with FECC and SIL. The used particle encoding should be also considered in fitness function se-

lection. The common encoding, used with MEPSO in this paper, is liable to undercluster. Therefore, for that particle encoding scheme, the above mentioned fitness functions with a tendency to overcluster are more beneficial than for MD-PSO, for which underclustering is not a big problem. Furthermore, if the properties of data are known, it can be used to help in the fitness function selection. However, commonly data properties are not known or one wants to use the same settings for several different datasets. In such cases, we recommend Xu, WB, or Dunn variant DU_{23} with FECC. They are consistently successful for different measures, algorithms, and data properties. Dunn variant DU_{23} with FECC is the most successful on the most difficult data distributions (noise and/or overlapping clusters), while WB with FECC was the winner also when tested on real datasets.

In this work, we have not considered CVIs based on different distance measures such as point symmetry distance [57] and comparative evaluations of different distance measures used in PSC belong to our future plans.

[1] S. Das, A. Abraham, and A. Konar, "Automatic kernel clustering with a multi-elitist particle swarm optimization algorithm," *Pattern Recognition Letters*, vol. 29, no. 5, pp. 688 – 699, 2008.

[2] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Fractional particle swarm optimization in multidimensional search space," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 40, no. 2, pp. 298–319, 2010.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of IEEE Int. Conf. on Neural Networks*, (Perth, Australia), pp. 1942–1948, 1995.

[4] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. of IEEE Congr. on Evolutionary Computation*, (Anchorage, Alaska, USA), pp. 69–73, 1998.

[5] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization - an overview," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[6] Z. Xinchao, "A perturbed particle swarm algorithm for numerical optimization," *Applied Soft Computing*, vol. 10, no. 1, pp. 119 – 124, 2010.

[7] Z.-H. Zhan, J. Zhang, Y. Li, and Y.-H. Shi, "Orthogonal learning particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 15, pp. 832–847, Dec. 2011.

[8] R. Mendes, *Population topologies and their inuence in particle swarm performance*. PhD thesis, Departamento de Informatica, Escola de Engenharia, Universidade do Minho, Portugal, 2004.

[9] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: simpler, maybe better," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 204–210, June 2004.

[10] B. Jarboui, M. Cheikh, P. Siarry, and A. Rebai, "Combinatorial particle swarm optimization (CPSO) for partitional clustering problem," *Applied Mathematics and Computation*, vol. 192, no. 2, pp. 337 – 345, 2007.

[11] M. Omran, A. Salman, and A. Engelbrecht, "Image classification using particle swarm optimization," in *Proc. of 4th Asia-Pacific conference on simulated evolution and learning*, (Singapore), pp. 370–374, 2002.

[12] M. Omran, A. Salman, and A. Engelbrecht, "Dynamic clustering using particle swarm optimization with application in image segmentation," *Pattern Analysis and Applications*, vol. 8, pp. 332–344, Feb. 2006.

[13] R. Xu, J. Xu, and D. Wunsch, "Clustering with differential evolution particle swarm optimization," in *Proc. of IEEE Congr. on Evolutionary Computation*, (Barcelona, Spain), pp. 1–8, Jul. 2010.

[14] R. Xu, J. Xu, and D. Wunsch, "A comparison study of validity indices on swarm-intelligence-based clustering," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 4, pp. 1243–1256, 2012.

[15] S. Rana, S. Jasola, and R. Kumar, "A review on particle swarm optimization algorithms and their applications to data clustering," *Artificial Intelligence Review*, vol. 35, pp. 211–222, Mar. 2011.

[16] A. A. Esmin, R. Coelho, and S. Matwin, "A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data," *Artificial Intelligence Review*, pp. 1–23, 2013.

[17] S. Kiranyaz, T. Ince, and M. Gabbouj, *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Germany: Springer Verlag Berlin Heidelberg, 2014.

[18] S. Kiranyaz, S. Uhlmann, T. Ince, and M. Gabbouj, "Perceptual dominant color extraction by multi-dimensional particle swarm optimization," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, p. 451638, 2009.

[19] T. Ince, S. Kiranyaz, and M. Gabbouj, "Evolutionary RBF classifier for polarimetric SAR images," *Expert Systems with Applications*, vol. 39, no. 5, pp. 4710 – 4717, 2012.

[20] J. Raitoharju, S. Kiranyaz, and M. Gabbouj, "Training radial basis function neural networks for classification via class-specific clustering," *IEEE Transactions on Neural Networks and Learning Systems*, In Press.

[21] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, Feb. 1956.

[22] K. Govindarajan, D. Boulanger, V. S. Kumar, and Kinshuk, "Parallel particle swarm optimization (ppso) clustering for learning analytics," in *Proc. of IEEE Int. Conf. on Big Data*, pp. 1461–1465, Oct 2015.

[23] C. Vimalarani, R. Subramanian, and S. N. Sivanandam, "An enhanced PSO-based clustering energy optimization algorithm for wireless sensor network," *The Scientific World Journal*, vol. 2016, 2016.

[24] L. Vendramin, P. A. Jaskowiak, and R. J. G. B. Campello, "On the combination of relative clustering validity criteria," in *Proc. of 25th Int. Conf. on Scientific and Statistical Database Management, ACM*, (Baltimore, MD, USA), pp. 1–12, Jul 2013.

[25] W. Sheng, S. Swift, L. Zhang, and X. Liu, "A weighted sum validity function for clustering with a hybrid niching genetic algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 35, no. 6, pp. 1156–1167, 2005.

[26] E. Dimitriadou, S. Dolniar, and A. Weingessel, "An examination of indexes for determining the number of clusters in binary data sets," *Psychometrika*, vol. 67, no. 1, pp. 137–159, 2002.

[27] A. Fujita, D. Y. Takahashi, and A. G. Patriota, "A non-parametric method to estimate the number of clusters," *Computational Statistics & Data Analysis*, vol. 73, pp. 27 – 39, 2014.

[28] S. Kiranyaz, T. Ince, J. Pulkkinen, and M. Gabbouj, "Classification of Holter registers by dynamic clustering using multi-dimensional particle swarm optimization," in *Proc. of Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society*, (Buenos Aires, Argentina), pp. 4695–4698, 2010.

[29] D. W. Van Der Merwe and A. Engelbrecht, "Data clustering using particle swarm optimization," in *Proc. of IEEE Congr. on Evolutionary Computation*, vol. 1, (Canberra, Australia), pp. 215–220, 2003.

[30] G. H. Ball and D. J. Hall, "ISODATA, a novel method of data analysis and pattern classification," tech. rep., Stanford Research Institute, Menlo Park, CA, USA, 1965.

[31] G. Milligan and M. Cooper, "An examination of procedures for determining the number of clusters in a data set," *Psychometrika*, vol. 50, no. 2, pp. 159–179, 1985.

[32] L. Vendramin, R. J. G. B. Campello, and E. R. Hruschka, "Relative clustering validity criteria: A comparative overview," *Statistical Analysis and Data Mining*, vol. 3, pp. 209–235, August 2010.

[33] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. USA: John Wiley & Sons, Inc., 2000.

[34] L. Xu, "Bayesian Ying-Yang machine, clustering and number of clusters," *Pattern Recognition Letters*, vol. 18, no. 11-13, pp. 1167 – 1178, 1997.

[35] Q. Zhao, M. Xu, and P. Fränti, "Sum-of-squares based cluster validity index and significance analysis," in *Adaptive and Natural Computing Algorithms* (M. Kolehmainen, P. Toivanen, and B. Beliczynski, eds.),

vol. 5495 of *Lecture Notes in Computer Science*, pp. 313–322, Germany: Springer Berlin Heidelberg, 2009.

[36] R. S. Hill, "A stopping rule for partitioning dendrograms," *Botanical Gazette*, vol. 141, no. 3, pp. pp. 321–324, 1980.

[37] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53 – 65, 1987.

[38] S. Bandyopadhyay and U. Maulik, "Nonparametric genetic clustering: comparison of validity indices," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 31, no. 1, pp. 120–125, 2001.

[39] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 224–227, 1979.

[40] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Perez, and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern Recognition*, vol. 46, no. 1, pp. 243 – 256, 2013.

[41] M. Kim and R. Ramakrishna, "New indices for cluster validity assessment," *Pattern Recognition Letters*, vol. 26, no. 15, pp. 2353 – 2363, 2005.

[42] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*. Boston, MA, USA: Academic Press, 4th ed., 2008.

[43] J. Bezdek and N. Pal, "Some new indexes of cluster validity," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 28, no. 3, pp. 301–315, 1998.

[44] C.-H. Chou, M.-C. Su, and E. Lai, "A new cluster validity measure and its application to image compression," *Pattern Analysis and Applications*, vol. 7, no. 2, pp. 205–220, 2004.

[45] J. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *Cybernetics and Systems*, vol. 3, no. 3, pp. 32–57, 1973.

[46] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On clustering validation techniques," *Journal of Intelligent Information Systems*, vol. 17, pp. 107–145, Dec. 2001.

[47] R. Tibshirani, G. Walther, and T. Hastie, "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, 2001.

[48] U. Maulik and S. Bandyopadhyay, "Performance evaluation of some clustering algorithms and validity indices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 12, pp. 1650–1654, 2002.

[49] I. Gurrutxaga, J. Muguerza, O. Arbelaitz, J. M. Perez, and J. I. Martin, "Towards a standard methodology to evaluate internal cluster validity indices," *Pattern Recognition Letters*, vol. 32, no. 3, pp. 505 – 515, 2011.

[50] P. Jaccard, "The distribution of the flora in the alpine zone," *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[51] D. Steinley, "Properties of the hubert-arabie adjusted rand index," *Psychological Methods*, vol. 9, no. 3, pp. 386–396, 2004.

[52] K. Bache and M. Lichman, "UCI machine learning repository," 2013.

[53] I. Guyon, U. V. Luxburg, and R. C. Williamson, "Clustering: Science or art," in *NIPS Workshop on Clustering Theory*, (Vancouver, Canada), 2009.

[54] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," in *Proceedings of the 7th International Conference on Evolutionary Programming VII*, EP '98, (London, UK), pp. 591–600, Springer-Verlag, 1998.

[55] A. Rezaee Jordehi and J. Jasni, "Parameter selection in particle swarm optimisation: a survey," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 25, no. 4, pp. 527–548, 2013.

[56] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, Dec. 2006.

[57] S. Bandyopadhyay and S. Saha, "A point symmetry-based clustering technique for automatic evolution of clusters," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, pp. 1441–1457, Nov 2008.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3903-9
ISSN 1459-2045