A Computational Intelligence Approach to Clustering of Temporal Data

by

Kristina Slavomirova Georgieva

Submitted in partial fulfilment of the requirements for the degree
Masters (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

July 2014

Publication data:

Kristina Slavomirova Georgieva. A Computational Intelligence Approach to Clustering of Temporal Data. Masters thesis, University of Pretoria, Department of Computer Science, Pretoria, South Africa, April 2014.

Electronic, hyperlinked versions of this thesis are available online, as Adobe PDF files, at:

http://cirg.cs.up.ac.za/

http://upetd.up.ac.za/UPeTD.htm

A Computational Intelligence Approach to Clustering of Temporal Data

by

Kristina Slavomirova Georgieva
E-mail: kristina.s.georgieva@gmail.com

Abstract

Temporal data is common in real-world datasets. Analysis of such data, for example by means of clustering algorithms, can be difficult due to its dynamic behaviour. There are various types of changes that may occur to clusters in a dataset. Firstly, data patterns can migrate between clusters, shrinking or expanding the clusters. Additionally, entire clusters may move around the search space. Lastly, clusters can split and merge.

Data clustering, which is the process of grouping similar objects, is one approach to determine relationships among data patterns, but data clustering approaches can face limitations when applied to temporal data, such as difficulty tracking the moving clusters. This research aims to analyse the ability of particle swarm optimisation (PSO) and differential evolution (DE) algorithms to cluster temporal data. These algorithms experience two weaknesses when applied to temporal data. The first weakness is the loss of diversity, which refers to the fact that the population of the algorithm converges, becoming less diverse and, therefore, limiting the algorithm's exploration capabilities. The second weakness, outdated memory, is only experienced by the PSO and refers to the previous personal best solutions found by the particles becoming obsolete as the environment changes. A data clustering algorithm that addresses these two weaknesses is necessary to cluster temporal data.

This research describes various adaptations of PSO and DE algorithms for the purpose of clustering temporal data. The algorithms proposed aim to address the loss of diversity and outdated memory problems experienced by PSO and DE algorithms. These problems are addressed by combining approaches previously used for the purpose of dealing

with temporal or dynamic data, such as repulsion and anti-convergence, with PSO and DE approaches used to cluster data. Six PSO algorithms are introduced in this research, namely the data clustering particle swarm optimisation (DCPSO), reinitialising data clustering particle swarm optimisation (RDCPSO), cooperative data clustering particle swarm optimisation (CDCPSO), multi-swarm data clustering particle swarm optimisation (MDCPSO), cooperative multi-swarm data clustering particle swarm optimisation (CMDCPSO), and elitist cooperative multi-swarm data clustering particle swarm optimisation (eCMDCPSO). Additionally, four DE algorithms are introduced, namely the data clustering differential evolution (DCDE), re-initialising data clustering differential evolution (RDCDE), dynamic data clustering differential evolution (DCDynDE), and cooperative dynamic data clustering differential evolution (CDCDynDE).

The PSO and DE algorithms introduced require prior knowledge of the total number of clusters in the dataset. The total number of clusters in a real-world dataset, however, is not always known. For this reason, the best performing PSO and best performing DE are compared. The CDCDynDE is selected as the winning algorithm, which is then adapted to determine the optimal number of clusters dynamically. The resulting algorithm is the k-independent cooperative data clustering differential evolution (KCDCDynDE) algorithm, which was compared against the local network neighbourhood artificial immune system (LNNAIS) algorithm, which is an artificial immune system (AIS) designed to cluster temporal data and determine the total number of clusters dynamically. It was determined that the KCDCDynDE performed the clustering task well for problems with frequently changing data, high-dimensions, and pattern and cluster data migration types.

Keywords: Clustering, Temporal Data, Data Mining, Computational Intelligence, Particle Swarm Optimization, Differential Evolution, Local Network Neighbourhood Artificial Immune System.

Supervisors : Prof. A. P. Engelbrecht
Department : Department of Computer Science
Degree : Master of Science

"The most exciting phrase to hear in science, the one that heralds the most discoveries, is not 'Eureka!' (I found it!) but 'That's funny...' "

- Isaac Asimov

"The only way to discover the limits of the possible is to go beyond them into the impossible"

- Arthur C. Clarke

Acknowledgements

I would like to express my sincere gratitude to the following institutions and people for the support provided during this research:

- Prof Andries P. Engelbrecht for the excellent mentorship and guidance as well as financial support.
- Darren Young, for the unceasing encouragement and patience.
- Marion Koleva for the inspiration.
- Barend Leonard and Filipe Nepomuceno for all the help regarding the cluster, CILib and F-race.
- Masters club for the valuable weekly inputs.
- Dr. Attie Graaff for the auto-generated temporal data clustering datasets and the LNNAIS results.
- SAP P&I BIT Mobile Empowerment for the unique opportunities and financial support. Opinions expressed in this thesis and conclusions arrived at, are those of the author and not necessarily to be attributed to SAP P&I BIT Mobile Empowerment.
- The National Research Foundation for the financial support. Opinions expressed in this thesis and conclusions arrived at, are those of the author and not necessarily to be attributed to the National Research Foundation.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

Large datasets are common in real-world applications. Due to the large volumes of data, it is difficult for humans to derive meaningful interpretations of the data patterns documented in these datasets. Computational intelligence (CI) [37], a branch of artificial intelligence (AI) that consists of adaptive algorithms, has introduced a variety of methods to aid the analysis of datasets [13][48][90][106].

Data clustering [56], which refers to the process of grouping similar objects together, is one method that can be used for data analysis. The overall aim of a data clustering algorithm is to determine similarities among data patterns and group the data patterns that are similar to each other, such that groups of data are well separated and compact. CI algorithms can be applied to data clustering, which may be used for classification, compression and interpretation of datasets [56]. Such research in CI algorithms for data clustering includes work by Hanuman *et al* on using differential evolution (DE) [51], Van der Merwe and Engelbrecht [118] on using particle swarm optimisation (PSO), and Graaff and Engelbrecht [46] on using artificial immune system (AIS), amongst others [75][79][103].

Datasets can include temporal data [92], which is data that changes with time. Such temporal data requires clustering algorithms to not only group similar data patterns, but also to track changes in the underlying clusters.

This thesis focuses on introducing CI algorithms to cluster temporal data. The research proposes a CI algorithm that determines the optimal number of clusters dynamically while clustering temporal data. The proposed algorithm is compared to an existing temporal data clustering algorithm. The rest of this chapter summarises the motivation, objectives, contributions of the thesis, and outlines the sections of the document.

1.1 Motivation

A number of temporal data clustering approaches have been previously developed [6][82] [113][122]. Examples of these include ST-DBSCAN [6] for spatio-temporal data clustering, the fuzzy time-series (FSTS) algorithm for clustering time-series data [82], the K-means algorithm for clustering time-series data [122], and a hidden Markov model (HMM)-based approach to temporal data clustering [89]. Few, however, are CI-based temporal data clustering algorithms. Examples of CI-based temporal data clustering algorithms include the local network neighbourhood artificial immune system (LNNAIS) [46], the Kohonen self-organisation map neural network (KSOM-NN) [120], a clustering self-organising map (SOM) [43], an artificial neural network (ANN) approach to clustering time-series data [127], and a combination of SOM and autoregressive integrated moving average (ARIMA) time-series models [119].

This research focuses on determining the ability of PSO and DE algorithms to cluster temporal datasets. These algorithms experience two weaknesses when applied to dynamic environments [9]. The first weakness, outdated memory, is only experienced by the PSO. The particles in a PSO hold the value of the best location in the search space found by each particle. When a change in the environment occurs, such as the data patterns within a dataset changing, the cluster centroids remembered by the particles become obsolete. The second weakness, loss of diversity, is experienced by both algorithms. As the entities in the population-based algorithms begin to converge to a solution, the diversity of the population decreases. The decrease in diversity results in a population with less exploration capabilities and, therefore, with less individuals capable of finding new solutions when a change in the environment occurs.

Real-world datasets have an unknown number of clusters. Although many data clustering algorithms exist, many require prior knowledge of the optimal number of clusters in the dataset. Automatically determining the optimal number of clusters is a common challenge in data clustering [56] and one that the algorithms need to address in order to be applied to real-world datasets where the total number of clusters is unknown.

In order to develop a temporal data clustering PSO or DE, appropriate approaches to dealing with the loss of diversity and outdated memory problems must be determined. Additionally, a method of identifying the optimal number of clusters dynamically must be put in place.

1.2 Objectives

This research aims to develop a temporal data clustering PSO or DE algorithm that requires no prior knowledge of the optimal number of clusters in the dataset. In order to reach this objective the following sub-objectives have been identified:

- Identify the limitations PSO and DE face when applied to dynamic environments.
- Conduct a survey of the techniques applied to algorithms to handle dynamic environments.
- Conduct a survey of the available PSO, DE and AIS data clustering techniques for static and temporal data.
- Conduct a survey of the techniques used to determine the optimal number of clusters dynamically.
- Identify measures used to determine the quality of a clustering solution.
- Select existing PSO algorithms designed for data clustering and dynamic environments.
- Implement and adapt selected PSO algorithms to cluster temporal data.

- Analyse the performance of the PSO algorithms over various frequencies of change, severities of change, dimensions and pattern migration types.
- Select existing DE algorithms designed for data clustering and dynamic environments.
- Implement and adapt selected DE algorithms to cluster temporal data.
- Analyse the performance of the DE algorithms over various frequencies of change, severities of change, dimensions and pattern migration types.
- Select the most effective PSO and most effective DE and analyse the performance of the two algorithms, selecting the best performing one of the two.
- Create an algorithm to cluster temporal data without the need for knowledge about the optimal number of clusters.
- Compare the performance of the proposed algorithm to that of the LNNAIS.

1.3 Contributions

The novel contributions of this work are:

- A number of adapted PSO algorithms for static and temporal data clustering.
- A number of adapted DE algorithms for static and temporal data clustering.
- A new DE algorithm that determines the optimal number of clusters dynamically and clusters temporal data.
- An analysis of the performance of the algorithms under various environmental change conditions.
- A comparison of the performance of the best PSO and DE algorithms developed, finding that the cooperative dynamic data clustering differential evolution (CDC-

DynDE) algorithm performed the clustering task most effectively.

- An analysis of the performance of the k-independent cooperative data clustering differential evolution (KCDCDynDE) and LNNAIS algorithms under various environmental change conditions.

- A comparison of the performance of the KCDCDynDE and the LNNAIS algorithms, finding that the KCDCDynDE performed better than the LNNAIS on data that was more frequently changing, of higher dimension, and comprised of pattern or cluster migration types.

1.4 Thesis Outline

This thesis consists of nine chapters, each dealing with a separate topic. The chapters are organised as follows:

- **Chapter 2** introduces the reader to the main concepts of the thesis. The chapter provides a background on CI and temporal data clustering. Additionally, it summaries previous work relevant to the discussed topics.

- **Chapter 3** introduces a variety of performance measures used to determine the quality of a clustering solution. The chapter concludes summarising the approaches chosen in this research.

- **Chapter 4** summarises the details of the artificial datasets used in this research.

- **Chapter 5** summarises the PSO algorithms considered and applied to temporal data clustering. The chapter introduces the algorithms that were adapted for temporal data clustering as well as the adaptations performed.

- **Chapter 6** summarises the DE algorithms considered and applied to temporal data clustering. The chapter introduces the algorithms that were adapted for temporal data clustering as well as the adaptations performed.

- **Chapter 7** summarises AIS algorithms and the LNNAIS.
- **Chapter 8** introduces the temporal data clustering algorithm derived from this research, which determines the optimal number of clusters dynamically.
- **Chapter 9** summarises and discusses the results of applying the discussed algorithms to the datasets introduced in Chapter 4.
- **Chapter 10** summarises the findings of this research.

The document contains three Appendices, which are organised as follows:

- **Appendix A** provides the optimised parameters per problem for the LNNAIS algorithm.
- **Appendix B** provides the optimised parameters per problem for the KCDC-DynDE algorithm.
- **Appendix C** provides an example XML of the KCDCDynDE for CILib.
- **Appendix C** provides a list of the important acronyms used or newly defined in the course of this work, as well as their associated definitions.
- **Appendix E** lists and defines the mathematical symbols used in this work, categorised according to the relevant chapter in which they appear.
- **Appendix F** contains the publications derived from this work.

Chapter 2

Literature Study

This chapter introduces the reader to the main concepts discussed in this research. These concepts include CI, data clustering, and temporal data. The challenges of clustering temporal data and previous work are also discussed in detail as the chapter progresses. There are three sections in this chapter, each dealing with a separate topic. The sections are organised as follows:

- Section 2.1 - **An Overview of Computational Intelligence**: This section summarises CI and the five CI paradigms.
- Section 2.2 - **Temporal Data Clustering**: This section summarises data clustering and the types of data clustering. The chapter proceeds to discuss temporal data, the types and uses of temporal data, and concludes with the use and problems of temporal data clustering.
- Section 2.3 - **Previous Work**: This section summarises previous research on CI algorithms applied to data clustering, dynamic data clustering and temporal data clustering.

2.1 An Overview of Computational Intelligence

CI [35][37] is a branch of AI which consists of a variety of adaptive algorithms that are capable of learning and adapting to new environments. These algorithms often model biological processes or the behaviour of living organisms in order to solve problems involving function optimisation, classification, clustering, pattern recognition, data mining and decision making. CI consists of five main paradigms [37] of CI, namely ANNs, AIS, swarm intelligence (SI), evolutionary computation (EC) and fuzzy systems (FS).

ANNs [8][37] model the connections between neurons in the biological brain. ANNs consist of a set of artificial neurons connected to each other to form a network. Each connection has a weight associated with it, representing the strength of the connection between the two connected neurons. The more commonly used architecture of an ANN consists of three layers where different calculations take place, namely an input layer connected to a hidden layer which is connected to an output layer. A neuron in the input layer is given a set of inputs which pass through the other layers, training the network to recognise future patterns. The training process involves changing the weight values associated with the connections between neurons of different layers, creating a network configured specifically to solve the problem at hand.

AISs [38] model the learning process of a biological immune system in order to recognise patterns. The algorithms consist of a set of artificial lymphocytes (ALC), which represent the fighter cells in the immune system. These ALCs react to each other as well as to antigen patterns, which are the foreign patterns that the AIS has to learn. The antigen patterns are exposed to the AIS, the affinities (binding strength) between ALCs and antigen patterns are calculated and the ALCs with the highest affinity values are adapted in order to begin learning the pattern. This adaptation may take place using negative selection [41], clonal selection [16][47], or some evolution-based technique [47][69][96]. Lastly, stimulation levels of each ALC are adapted by making resourceful ALCs have more influence on the population than the less resourceful ones.

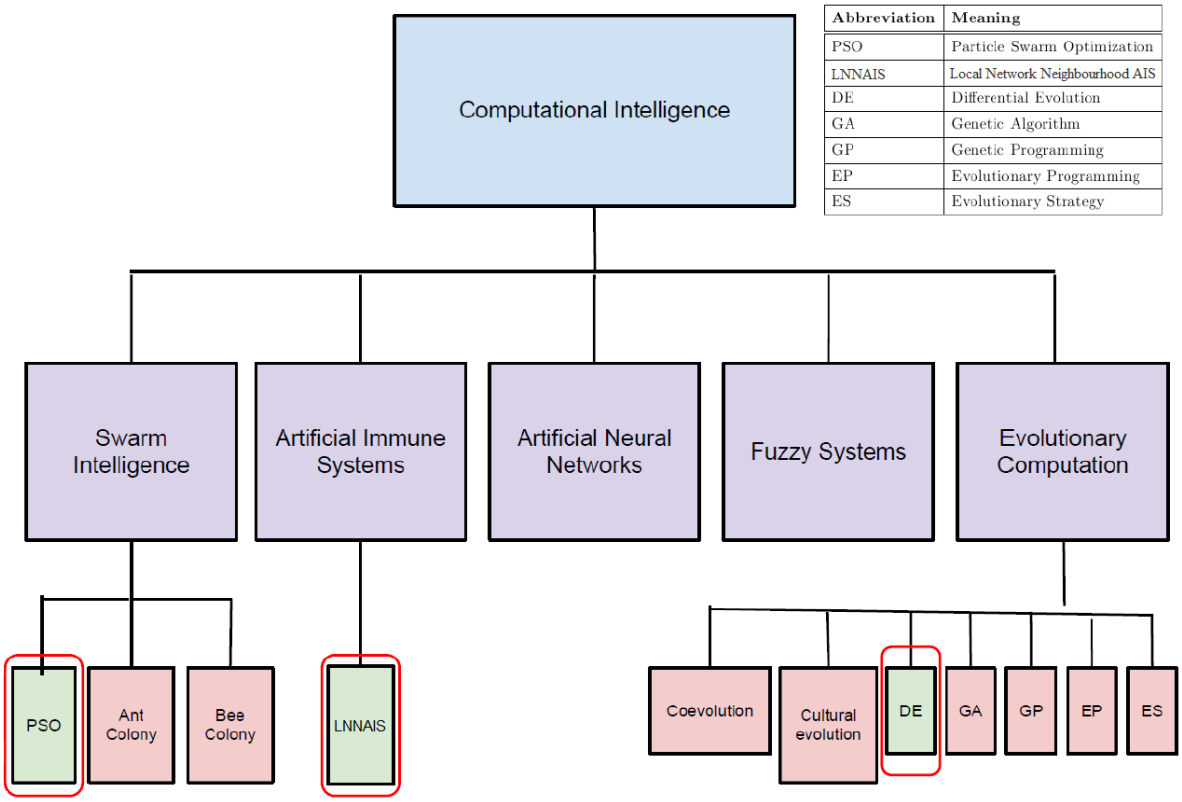
SI [14][68] algorithms model the collective behaviour of self-organising insects, animals and organisms. They consist of a population of individuals, which represent these organ-

isms, animals or insects, spread across the search space of the defined problem. These individuals explore the search space and socially influence each other with the goal of swarming around the best solution to a problem. The individuals move around the search space in search of good solutions and influencing each other's position, resulting in a swarm of individuals surrounding a potential solution to a problem.

EC [37][40] models the evolutionary process described by Darwin [24], Lamarck [71] and Spencer's [107] survival of the fittest theory [87][126]. The algorithms consist of a population of individuals spread across the search space of a problem. These individuals can evolve through selection, crossover and mutation, mimicking the genetic outcome of mating organisms and genetic mutation. A number of individuals are selected and recombined, creating offspring with elements from each of the parents. The offspring are then mutated to add genetic variety to the population. A selection strategy is used to choose which individuals from the parent and offspring populations survive to the next generation, mimicking survival of the fittest. The evolutionary processes used and the order in which they are applied vary across algorithms.

FSs [35][37][134] model human reasoning through the use of fuzzy sets. Fuzzy sets, introduced by Lotfi Zadeh [135] are sets where elements of the sets have different degrees of membership instead of binary membership (belonging or not belonging). By using fuzzy sets, inferencing on the FSs represents human reasoning and real world problems more accurately as there is a level of uncertainty involved. Some algorithms may be the fuzzy adaptation of an existing non-fuzzy algorithm. Such an adaptation would require changing the algorithm from handling binary membership to handling fuzzy membership.

This research focuses on three of the CI paradigms, namely SI, EC and AIS, as illustrated in Figure 2.1. The algorithms discussed are PSO, DE and LNNAIS. Variations of these algorithms are implemented and analysed on various temporal clustering problems. The best performing algorithm is adapted to determine the number of clusters automatically.



Abbreviation	Meaning
PSO	Particle Swarm Optimization
LNNAIS	Local Network Neighbourhood AIS
DE	Differential Evolution
GA	Genetic Algorithm
GP	Genetic Programming
EP	Evolutionary Programming
ES	Evolutionary Strategy

Figure 2.1: Computational Intelligence Paradigms with highlighted algorithms to be tested

2.2 Temporal Data Clustering

Data that changes through time can be found in geospatial [123], business [124], transportation [139] and archaeological [114] applications. Data clustering is a process that can be used to determine relationships among data patterns [57]. Tracking these relationships as they change may reveal interesting time-related information about the dataset being clustered. This section defines data clustering, temporal data, and summarises the clustering of temporal data.

2.2.1 An Overview on Data Clustering

Clustering [56] refers to the process of grouping similar objects together. In CI, these objects are referred to as data patterns. The term data clustering was first mentioned by Clements [21], where data clustering was described with regards to anthropological data [56][105]. Since then, many clustering algorithms have been created and existing algorithms adapted for the purpose of grouping similar data.

There are two main classes of data clustering algorithms [56], namely hierarchical and partitional clustering. Hierarchical clustering [138] involves generating a hierarchical tree by initially assigning each data pattern to its own cluster. These clusters are the tree's leaf nodes. The most similar pairs of clusters are then merged, creating a set of clusters with two data patterns each. These clusters are then the parent nodes of the two merged clusters. The process continues until everything is in one cluster, which is the root node of the tree, leading to a resulting tree which contains different levels of clusters. The construction of the tree can be done in reverse by starting at the root node and dividing the data into clusters until there is one data pattern per cluster at the leaf nodes.

Unlike hierarchical clustering, partitional clustering [94] does not result in a hierarchical tree. Partitional clustering involves partitioning data simultaneously by attempting to increase the similarities between data patterns within one group and decreasing similarities between data patterns in different groups. This results in a set of clusters containing similar data patterns, not in a tree with different levels of clusters. The three algorithms discussed in this research are partitional clustering methods.

Clustering can be used for various purposes [56], namely classification, gaining insight of the data being analysed and compression. Clustering for classification consists of data patterns being clustered together to form part of one class, where each resulting cluster represents a different class. Another use of clustering is to gain insight into data being analysed, illustrating similar data patterns, differing data patterns and outliers. Lastly, clustering can be used for compression, where similar data is grouped and represented as one summarised data pattern.

2.2.2 Temporal Data

Temporal data [92] refers to data which is time-stamped due to time being of high importance to a data pattern's meaning. This data can be divided into four categories, as described by Krishna and Roy [70], namely static data, sequence data, time-stamped data and fully temporal data. Static data itself has no reference to time nor does its analysis lead to temporally relevant results. Sequence data, on the other hand, refers to data items which have an implicit temporal relationship between each other. Alternatively, an explicit temporal relationship between data items is encountered in time-stamped data. This differs from fully temporal data as fully-temporal data is time-stamped data where the validity of the data is dependant on time.

Temporal data is encountered in various fields. In archaeology, for example, spatio-temporal data is very prominent due to the wide use of geographical information system (GIS) [125]. Spatio-temporal data can also be encountered in forest fire forecasting [18], weather forecasting [131], traffic management [131], vegetation change analysis [54], vegetation degradation analysis [54] and military applications [61]. Temporal data does not, however, need to have a spatial component. Examples of applications of temporal data can be found in archaeology [81], medicine [19], military [60], and stock market applications [78].

2.2.3 Clustering of Temporal Data

Data clustering, as discussed previously, is the process of grouping similar data patterns together. In a static dataset, data patterns need be clustered once and the resulting clusters are analysed. When dealing with temporal data, however, data patterns change with time and therefore need to be clustered for each time frame [56]. The changing patterns cause the clusters to differ between time frames. These clusters do not only change position, they can change composition, merge and split [61]. Patterns from one cluster may move to another cluster, increasing or decreasing the size of a cluster. Entire clusters may move as a whole, keeping the same number of patterns. Lastly, clusters

may appear and disappear as a result of patterns migrating between different clusters, being added to or being removed from the clusters.

Two problems are introduced by temporal data when performing the clustering task using the CI algorithms discussed in this research. These problems are the loss of diversity in the population and outdated memory of entities in the population [9].

The three population based CI algorithms which are discussed in this research, namely PSO [66], AIS [38], and DE [109], use a population of individuals which is spread across the search space of a particular problem. This causes the population to be diverse after initialisation because individuals are all placed at random positions around the search space. As the algorithm proceeds searching for a solution, these individuals slowly gather around the solution, decreasing diversity. When the algorithm finds a solution, the diversity of the population is very low, because individuals have converged to this solution. In a temporary environment this decrease in diversity is problematic because, after an environment change takes place, no individuals are exploring the search space to attract the population towards the new solution [15].

PSO algorithms have a memory system where previous positions or values of individuals are kept by the individuals themselves. As an environment changes, this memory becomes obsolete. Positions and values of individuals change making previous findings inaccurate and inapplicable to the new environment [9]. This will cause individuals to be influenced by the obsolete positions, potentially missing the new solutions during the search.

2.3 Previous Work

Data clustering is a commonly used method for classification and relationship discovery. Different types of data, such as images [76], spatial data [100], categorical values [55], numeric values [55], and text [116] have been clustered by various algorithms. This data can be static or dynamic.

Xu and Wunsch [130] performed a survey on clustering algorithms in 2005. The research applied K-means, fuzzy C-means, hierarchical clustering, CLARA, CLARANS, BIRCH, DBSCAN, CURE, WaveCluster, DENCLUE, FC, CLIQUE, OptiGrid and ORCLUS on the Iris and Mushroom benchmark datasets [5]. Xu and Wunsch [130] did, however, not test any CI algorithms. Van der Merwe and Engelbrecht [118], on the other hand, used a PSO algorithm for clustering, Liu *et al* [74] and Graaff and Engelbrecht [47] developed a clustering AISs, Das *et al* [26] adapted a DE algorithm for data clustering and Herrero *et al* [53] used an ANN to cluster gene expression patterns. These researchers have all used CI algorithms for clustering various types of data.

When dealing with temporal data, the data needs firstly to be represented appropriately in order for the algorithm to easily process the input. The data can be either kept in its original form or represented as continuous or discrete adaptations of the original set [3]. Discretization, however, has the disadvantage of possibly causing loss of information if used inappropriately [39]. Ideally, the original data, after eliminating irrelevant and missing values, is used in order to not loose important information during transformation. Some formats may, however, be difficult for the algorithm to process and interpret causing the use of the original data to often be a less viable option. The CI algorithms discussed in this paper process numeric data, requiring for non-numeric values to be transformed or eliminated.

Researchers have transformed various types of data to numeric data in order for CI algorithms to easily process the input. Das *et al* [26], for example, represented grayscale images as values of pixel intensities in order for them to be processed and clustered using DE and PSO algorithms. Another example is the use of discrete values mentioned by Johnson and Johnson [59], where numeric values were used to represent discrete elements when analysing the Kansas City Hopewell ceramics dataset using the K-means algorithm.

In order for the data being processed to be temporal, it needs to be time-stamped, either with the time the record was added to the dataset or with a time relevant to the entry itself [112]. In an archaeological dataset of artefacts, for example, the former time refers to the time that the record describing the artefact was added to the dataset, while the latter refers to the time in history that the artefact belongs to. This time-stamping allows

algorithms to handle the time dependencies between data patterns in the datasets being analysed.

CI algorithms have been previously adapted to handle dynamic data, which behaves in the same manner as temporal data. In order to cluster the changing data patterns, algorithms need to track clusters through time. The three algorithms discussed in this paper are all population based CI algorithms with the aim of initially exploring the search space by having a population that consists of individuals in various locations. The population's diversity decreases and the algorithm results in a population of individuals located around an optimum. In order to track changes in dynamic datasets, the population of each algorithm needs to remain diverse and search for new solutions at all times.

The three algorithms that are discussed in this paper, PSO, DE and AIS are shortly introduced below. Each introduction is followed by a discussion of previously developed adaptations of the algorithms for data clustering.

2.3.1 Particle Swarm Optimisation for Data Clustering

The PSO algorithm [33][66] is a population-based algorithm that models the social behaviour of flocking birds. A population is initialised randomly within the search space. Each individual, called a particle, has a velocity which is dependant on the best position the particle has found so far, the particle's previous velocity, the best position found by the particle's entire neighbourhood so far, two random components and two parameters determining the social and personal influence on the particle's velocity. The velocities of each particle are updated each iteration and the position of the particle is changed by adding the velocity to the particle's current position. The best position found by the particle is updated if the current position is better than the current best position. Particles move around the search space exploring for new solutions with the aim of swarming around the best position found.

In dynamic environments, PSO algorithms face two main problems tracking changes

in the environment [9]. The first problem is that of outdated memory. This refers to the fact that personal best positions remembered by the algorithm may no longer be applicable to the search space after the environment changes. The particle's velocity may, therefore, be influenced incorrectly by this outdated data. The second problem is that of diversity loss. This refers to the fact that particles converge and stop exploring the search space, making it more difficult to track a moving optimum.

In order to handle loss of memory, a PSO can, as soon as it detects a change, reset all personal best positions to the current particle positions [9]. The challenge is then to determine how the PSO detects a change. One solution is to, when a change occurs, re-evaluate the environment at the personal best positions and determine whether the personal best position's fitness has changed. If the fitness is the same, then no change has occurred, otherwise the particle's best position is changed to the current best position [9] as a change has taken place.

The loss of diversity can be addressed in a variety of ways. One solution is to use repulsion [9]. This involves particles being moved away from other particles or an already detected optimum through a repulsive force. In other words, as soon as a particle is within a certain radius of another particle (or detected optimum), the particle's velocity is changed with the aim of setting the particle's new position to one away from the other particle or optimum. Another solution is to use multiple swarms [10]. The multiple-swarm approach is favourable if many solutions must be found within the search space, such as multiple clusters (each population can search for an optimum). The aim is for each sub-swarm to converge around a different solution. Each sub-swarm is separate from another and remains separate through repulsion. Multi-swarm approaches can also increase exploration even further by adding an anti-convergence operator [11], ensuring that there is always one free swarm exploring the search space instead of converging to an optimum. Diversity can also be maintained using charged and neutral sub-swarms [12] as well as quantum particles [110][133].

Van der Merwe and Engelbrecht [118] developed a data clustering PSO, where a particle is represented by a group of centroid positions and the fitness of a particle is calculated by using the dataset. Yang *et al* [132] used the same particle representation as Van

der Merwe and Engelbrecht [118] and proposed a data clustering PSO that uses the K-harmonic means algorithm to refine the particles optimised by the PSO. Chen and Ye [17], on the other hand, developed a data clustering PSO where the particle is a vector representing centroid positions consecutively. Lastly, Cohen and De Castro [23] proposed a data clustering PSO where each particle represents a single possible centroid, and the entire population is required for the final clustering solution.

Li and Yang [73] developed a PSO algorithm aimed at clustering dynamic optimisation problems. The algorithm uses a hierarchical clustering approach for creating temporary sub-swarms, which are then further refined, requiring the algorithm to analyse the same data a second time. A local search is then performed on each refined sub-swarm and the sub-swarms are checked for overlap. Although the algorithm was effective for the analysed problems, it is unnecessarily computationally expensive.

Omran *et al* [85] developed the dynamic clustering PSO. This algorithm begins by randomly selecting a set of centroids from the available dataset and randomly initialising a swarm of particles. A binary PSO is then applied in order to find the best centroid positions, followed by the application of K-means for refining the selection. The refined set of best centroid positions is lastly added to a randomly selected set of centroid positions from the dataset and this process is repeated for the newly created set of cluster centroids.

2.3.2 Differential Evolution for Data Clustering

DE [109] algorithms are population-based evolutionary algorithms. The algorithms begin by randomly initialising a population of individuals, called chromosomes. A trial chromosome is generated for each chromosome by adding the scaled difference of two or more randomly selected chromosomes to the another selected chromosome. Crossover is then performed for each dimension of the trial chromosome by either replacing a selected dimension of the trial chromosome with the original value of the chromosome, or keeping the trial chromosome's value. The trial chromosome is lastly compared to the original chromosome and the chromosome with the best fitness is chosen to survive to the next

generation.

Mendes and Mohais [80] developed a DE algorithm, called dynamic differential evolution (DynDE), which uses multiple populations and exclusion for the purpose of solving dynamic optimisation problems. First, DynDE was designed to include quantum individuals, which are individuals that are generated around the global best position of the current sub-population and follow different rules to those of the standard individuals. The algorithm was then tested using Brownian individuals instead, which are individuals around the best position, generated by randomly adding Gaussian noise to the position of the best individual. This experiment was followed by using entropies to increase diversity, which involves adding a random value to each dimension of a surviving individual if a change in the environment is detected. The paper concluded that Brownian individuals are less computationally demanding than quantum individuals and better performing than entropies, and should therefore be favoured for the DynDE.

Du Plessis and Engelbrecht [30] analysed the DynDE algorithm developed by Mendes and Mohais [80] and proposed two improvements to the DynDE. The first proposal was to favour sub-populations. This involves allowing all sub-populations to evolve, and then giving stronger sub-populations more time to optimise, while the weaker ones are kept constant for a period of time. The second proposal was to add migrating individuals. This approach involves individuals from stronger populations migrating to weaker ones. It was, however, found to be unsuccessful and a different approach was proposed. This new approach required sub-populations to have set sizes for a certain number of generations. After some time, these sub-population sizes are changed to keep stronger sub-populations larger and weaker ones smaller, removing the weaker individuals and adding new ones in the same manner as generating Brownian individuals. Alternatively, a combination of both approaches, i.e. favoured sub-populations and migrating individuals, could be used. The paper concluded that the favoured sub-population approach did not show any improvement to the DynDE, and the migrating individuals approach, although good with higher change periods, was not as successful with lower change periods. Lastly, the combination approach displayed the opposite results to the migrating individuals by performing better in lower change periods than higher ones.

Omran and Engelbrecht [86] adapted the bare-bones DE to cluster data by changing the individual to represent a group of centroid positions and using the dataset in the fitness calculation of an individual. Additionally, Paterlini and Krink [93] used a DE, a PSO and a genetic algorithm (GA) for partitional clustering and concluded that the DE had the best performance when compared to the other two algorithms.

Das *et al* [26] proposed the automatic clustering differential evolution (ACDE) algorithm, which aims to dynamically determine the number of clusters in a static environment. Although the algorithm does attempt to retain diversity by adapting the scaling factor of the original DE algorithm, it was not developed for, nor tested in dynamic environments. The representation used by ACDE consists of a vector with two parts, the cluster centroids and the activation thresholds. An activation threshold is a floating-point value corresponding to a cluster centroid, which controls whether that centroid is activated or not. This may cause implementation to be highly index dependant and complex, as each activation threshold corresponds to a cluster centroid in a particular index of the chromosome.

2.3.3 Artificial Immune System for Data Clustering

AISs [38][37] are population-based models of the human immune system. The basic algorithm begins by randomly initialising a population of ALCs. A training set of antigen patterns, which are the patterns to be learned by the AIS, is chosen. For each antigen pattern in the training set, a sub-set of ALCs is chosen and compared to the antigen pattern. This results in the calculation of the antigen affinity of the ALC and antigen pattern, which refers to the binding strength between the two. The ALCs with the highest antigen affinity values are then adapted using a selection strategy. Lastly, the stimulation of each ALC is updated in order for ALCs to influence other ALC's reactions to foreign cells.

Although not developed for clustering, AISs have been used to cluster data. Liu *et al* [74] developed an algorithm where antigen patterns represent clusters and ALCs represent cluster centres. The algorithm was not developed to work in dynamic environments

and requires prior knowledge of the optimal number of clusters needed. The algorithm performed better than the K-means and a GA when applied to two static datasets, one generated by the authors and the other Anderson's Iris dataset [5].

Nasraoui *et al* [83] developed a scalable AIS model aimed at solving problems in dynamic environments with lower computational costs. The researchers approached the dynamic problem with an AIS algorithm due to the ease of adaptation to dynamic environments displayed by this type of algorithm. The algorithm was adapted by adding a number of improvements, including dynamic stimulation and suppression, compression, dendritic injection and adapting the cloning method of an AIS. Dynamic stimulation and suppression involve allowing ALCs to sustain themselves even after the antigen responsible for their creation disappears. In order to avoid adding to the size of the AIS, a limit to the memory of each ALC is imposed by decreasing the stimulation coefficient with age, and eventually removing outdated ALCs, unless the ALCs are stimulated by a new antigen pattern. Compression refers to the clustering of ALCs so that, as the algorithm progresses, only a subset of ALCs is used, optimising the algorithm. Dendritic injection forces an antigen to be duplicated if it fails to activate a minimum portion of the ALC network. This encourages more exploration as it adds new antigen patterns to the system. Lastly, the cloning process is adapted by not allowing cells to clone unless they are mature. The algorithm succeeded in the tasks presented to it and showed good adaptive characteristics due to the lack of diversity loss.

Graaff and Engelbrecht [47] proposed an AIS algorithm inspired by network theory called the LNNAIS. Like other network models, LNNAIS is a population based AIS where a network affinity value is calculated to represent the binding strength between groups of ALCs called networks. Unlike other network models, LNNAIS does not hold a network threshold parameter which determines whether two ALCs can form a network. LNNAIS creates index based neighbourhoods instead, which are determined by a neighbourhood radius. The neighbourhood is then updated by first adapting the ALC causing a change and then adapting the ALC's neighbours. The algorithm showed promising results when adapting patterns to the artificially generated dynamic environment.

2.3.4 Dynamic Determination of the Number of Clusters

The automatic determination of the optimal number of clusters K is a common problem in data clustering [56]. One common approach is to run the algorithm for various values of K and measuring the results using a validity index [20][29][110]. The K with the best resulting validity index value is then selected. This approach is, however, not ideal as it requires the algorithm to be ran a number of times. Another method was proposed by Omran *et al* [85], where a combination of a PSO and K-means was used in order to determine the number of clusters dynamically.

Splitting and merging of clusters as the algorithm progresses is another popular method of determining K dynamically. Wu *et al* [128] proposed an algorithm that uses the one-prototype-take-one-cluster paradigm, estimating the optimal number of clusters using splitting (driven by a distortion threshold) and merging when two clusters are close to each other. Li and Yang [73] proposed a clustering PSO for dynamic optimization which uses a merging hierarchical method to track K and multiple swarms to perform the data clustering. Veenman *et al* [121] proposed a clustering algorithm that dynamically determines K by minimising a cluster validity index through splitting and merging. Sun *et al* [110] used a fuzzy C-means splitting and proposed a new validity index to determine the number of clusters dynamically.

Frigui and Krishnapuram [42] proposed clustering by competitive agglomeration. This method combines hierarchical clustering with partitional clustering by partitioning the dataset and eliminating clusters that lose the competition for data points. Lee and Antonsson [72] proposed an evolutionary strategy for dynamic determination of the number of clusters, where the crossover operation allows for the addition and removal of clusters. Graaff [46] proposed an AIS algorithm, where ALCs multiply through cloning and are eliminated through merging, resulting in ALC networks representing clustering solutions.

2.4 Summary

This chapter introduced the reader to the main concepts needed for this research. The chapter began by describing CI and its five paradigms, which was followed by an overview of data clustering, temporal data and temporal data clustering. The chapter concluded by summarising previous work done on temporal data clustering.

Chapter 3

Performance Measures

This chapter introduces a number of measures used to determine the quality of a clustering solution at any point in time. The chapter summarises measures to determine the level of separation and compactness of the clusters. Additionally, measures to determine the quality of a final clustering solution are introduced, followed by fitness functions previously used to guide the data clustering process. The chapter concludes by defining an approach to using the measures selected for this research for dynamic environments.

There are four sections in this chapter, each dealing with a separate topic. The sections are organised as follows:

- Section 3.1 - **Inter-cluster Distance**: This section summarises the inter-cluster distance measure.
- Section 3.2 - **Intra-cluster Distance**: This section summarises the intra-cluster distance measure.
- Section 3.3 - **Validity Indices**: This section summarises various validity indices used to determine the quality of a clustering solution.
- Section 3.4 - **Fitness**: This section summarises various fitness measures used in data clustering.

3.1 Inter-cluster Distance

The inter-cluster distance measures the separability of a group of cluster centroids. The measure determines the average distance between different clusters by measuring the distance between the centroids of each cluster [46]. A centroid refers to the center position of a cluster [118]. The inter-cluster distance is measured using

$$J_{iter} = \frac{2}{K(K-1)} \sum_{k=1}^{K-1} \sum_{k_2=k+1}^{K} d(\mathbf{c}_k, \mathbf{c}_{k_2}) \tag{3.1}$$

where $d(\mathbf{c}_k, \mathbf{c}_{k_2})$ is the Euclidean distance between cluster centroid $\mathbf{c}_k$ and cluster centroid $\mathbf{c}_{k_2}$, calculated using

$$d(\mathbf{x}_a, \mathbf{x}_b) = \sqrt{\sum_{n=0}^{N} (x_{a_n} - x_{b_n})^2} \tag{3.2}$$

where $\mathbf{x}_a$ and $\mathbf{x}_b$ are the vectors on which the distance is being calculated, and N is the dimensionality of the vector.

3.2 Intra-cluster Distance

The intra-cluster distance [46] measures the average distance between data patterns and the centroid of the cluster that they belong to, using

$$J_{intra} = \frac{\sum_{k=1}^{K} \sum_{\forall \mathbf{z} \in C_k} d(\mathbf{z}_p, \mathbf{c}_k)}{|P|} \tag{3.3}$$

where C_k is a cluster, $\mathbf{z}_p$ is a data pattern and $|P|$ is the total number of data patterns in the dataset.

3.3 Validity Indices

Validity indices [49] are quantitative measures of the quality of a clustering solution. Various validity indices have been used in the past, examples of which are presented below.

Dunn Validity Index [32], which is a value that is maximised, meaning that a larger value means a better clustering solution, identifying how separated and compact clusters are [46]. This value is calculated using

$$Q_D = \min_{k=1,\ldots,K} \left\{ \min_{j=k+1,\ldots,K} \left\{ \frac{\min\limits_{\forall \mathbf{z}_1 \in C_k, \forall \mathbf{z}_2 \in C_j} \{d(\mathbf{z}_1, \mathbf{z}_2)\}}{\max\limits_{k=1,\ldots,K} \left\{ \max\limits_{\forall \mathbf{z}_1, \mathbf{z}_2 \in C_k} d(\mathbf{z}_1, \mathbf{z}_2) \right\}} \right\} \right\} \tag{3.4}$$

Davies Bouldin Validity Index [27], which measures the average similarity between two similar clusters [46], calculated using

$$Q_D = \frac{1}{K} \sum_{k=1}^{K} \max_{\substack{j=1,\ldots K \\ j \neq k}} \left\{ \frac{\frac{1}{2}\upsilon(C_k) + \frac{1}{2}\upsilon(C_j)}{d(\mathbf{c}_k, \mathbf{c}_j)} \right\} \tag{3.5}$$

where a smaller value represents a better clustering solution and the quality $\upsilon(C_k)$ of cluster C_k is computed as

$$\upsilon(C_k) = 2 \left[\frac{\sum\limits_{\forall \mathbf{z} \in C_k} d(\mathbf{z}_p, \mathbf{c}_k)}{|C_k|} \right] \tag{3.6}$$

where $|C_k|$ is the total number of data patterns in cluster C_k.

Ray-Turi Validity Index [99], which is a measure that combines the intra-cluster and inter-cluster distance measures into one using

$$Q_D = \frac{J_{intra}}{inter_{min}} \tag{3.7}$$

where J_{intra} is the intra-cluster distance defined in equation (3.3), $inter_{min}$ is the smallest inter-cluster distance found using equation (3.1). A smaller value for Q_D represents a better clustering solution.

Xie-Benni Validity Index [129], which is a validity index specifically designed for fuzzy clustering problems, which measures the ratio of compactness to separation using

$$Q_D = \frac{\sum_{k=1}^{K}\sum_{p=1}^{|P|} u_{kp}^{m}||d(\mathbf{z}_p, \mathbf{c}_k)||^2}{n \min_{k\neq p}||d(\mathbf{c}_k, \mathbf{c}_p)||^2} \tag{3.8}$$

where u_{kp}^{m} is the fuzzy membership of a pattern to a cluster. A smaller value represents a better clustering solution.

Silhouette coefficient [62], which measures the number of data patterns that fit well within a cluster and the number of data patterns that do not, in order to quantify how good a clustering solution is. The silhouette of a data pattern $\mathbf{z}_p$ is calculated using [2]

$$s(\mathbf{z}_p) = \frac{b(\mathbf{z}_p) - a(\mathbf{z}_p)}{max\left\{a(\mathbf{z}_p), b(\mathbf{z}_p)\right\}} \tag{3.9}$$

where $b(\mathbf{z}_p)$ is the average dissimilarity between data pattern $\mathbf{z}_p$ and all data patterns in the cluster closest to the cluster that $\mathbf{z}_p$ belongs to, and $a(\mathbf{z}_p)$ is the average dissimilarity between data pattern $\mathbf{z}_p$ and all other data patterns in the cluster that $\mathbf{z}_p$ belongs to. The dissimilarity measure can be any form of distance measure, such the Euclidean distance measure given in equation (3.2). After the silhouette of each individual has been calculated, the average silhouette is calculated to determine the quality of a clustering solution.

The silhouette coefficient is a value that a clustering algorithm aims to maximise, which means that higher values represent better clustering solutions. Kaufman and Rousseeuw

[62] provided a table showing an interpretation of what the possible values of the silhouette coefficient mean. Table 3.1 [62] shows these values, where 1.00 is the maximum possible value for the silhouette coefficient.

Table 3.1: Silhouette Coefficient Values

Value	**Interpretation**
0.71-1.00	A strong clustering structure has been found
0.51-0.70	A reasonable clustering structure has been found
0.26-0.50	A weak clustering structure has been found
≤ 0.25	No substantial structure has been found

Multi-objective optimisation has also been used for determining the quality of a clustering solution [50][102][111]. This involves two or more validity indices being optimised simultaneously using a multi-objective algorithm.

The validity index selected for this thesis is the Ray-Turi validity index as it is one of the more often used validity indices in literature and, therefore, allowed for results to be compared to those of Graaff [46].

3.4 Fitness

In general, the fitness of a candidate solution refers to the quality of that solution with respect to the objective being optimized [58]. The fitness is used to compare solutions to one another in order to determine which solution is better. Different problems require different fitness functions. For data clustering, a number of fitness measures have been used, including the quantization error [118], squared error [57], heuristic mean squared

error (MSE) modification [72], Xie-Benni validity index adaptation [1], and K-harmonic means objective function [137].

The quantization error [118] measures the *average* distance, for all clusters, between patterns in a cluster and the centroid of that cluster. The quantization error is defined as

$$J_e = \frac{\sum_{k=1}^{K} \frac{\sum_{\forall \mathbf{z}_p \in C_k} d(\mathbf{z}_p, \mathbf{c}_k)}{|C_k|}}{K} \tag{3.10}$$

The squared error [57][72] measures the *total* squared distance, for all clusters, between patterns in a cluster and the centroid of that cluster. The squared error is calculated using

$$SE = \sum_{k=1}^{K} \sum_{p=1}^{C_k} d(\mathbf{x}_k, \mathbf{z}_p)^2 \tag{3.11}$$

Lee and Antonsson [72] chose to use the heuristic MSE measure modification of the squared error to penalise solutions with a large number of clusters by changing equation (3.11) to

$$hMSE = \sqrt{K+1} \sum_{k=1}^{K} \sum_{p=1}^{C_k} d(\mathbf{c}_k, \mathbf{z}_p)^2 \tag{3.12}$$

Abraham *et al* [1] based their fitness measure on the Xie-Benni validity index, using

$$f = \frac{1}{XB_i(c) + \epsilon} \tag{3.13}$$

where $XB_i(c)$ is the Xie-Benni validity index calculated using equation (3.8) and ϵ is a very small constant greater than zero used to prevent division by zero.

Zhang *et al* [137] used a fitness function for the K-harmonic means algorithm which is defined as

$$f = \sum_{p=1}^{P} \frac{K}{\sum_{k=1}^{K} \frac{1}{||d(\mathbf{c}_k, \mathbf{z}_p)||^2}} \tag{3.14}$$

The fitness measure used in this thesis is the quantization error. This was selected in order to be able to compare final results to those of Graaff [46]. Equation (3.10) does however allow for division by zero if patterns have not been assigned to a centroid, which can occur if a centroid is initialized too far away from the data patterns. This problem can be remedied by initializing centroids to the positions of data patterns or to random positions within the bounds of the dataset. The latter would reduce the possibility of a division by zero, but note that it does not remove the problem completely. The solution used in this thesis is to compute the quantization error using

$$J_e = \begin{cases} \frac{\sum_{k=1}^{K} \frac{\sum_{\forall \mathbf{z}_p \in C_k} d(\mathbf{z}_p, \mathbf{c}_k)}{|C_k|}}{K} & \text{if } |C_k| > 0 \\ \infty & \text{if } |C_k| = 0 \end{cases} \tag{3.15}$$

where a bad fitness of ∞ is given to a solution with centroids far from data patterns, so that the bad solutions are filtered out by the algorithm, which is minimizing the quantization error.

The measures defined in this chapter are measures used in static environments. For dynamic environments each measure is calculated at each time-step of the algorithm. Once the algorithm has completed the search, the values are averaged over all time-steps and the total number of runs in order to obtain an accurate representation of the clustering solution in a dynamic environment [46].

The measure selected in this research to guide the algorithm's search for a solution is the quantization error. Furthermore, the silhouette coefficient is used as a condition for one of the algorithms proposed. Lastly, the inter-cluster distance, intra-cluster distance and Ray-Turi validity index are used to analyse the quality of a final solution.

3.5 Summary

This chapter covered a number of cluster quality measures previously used to analyse the solutions to data clustering problems. The chapter defines the quantization error, squared error, heuristic MSE, Xie-Benni validity index adaptation and K-harmonic means objective function fitness measures. The inter-cluster and intra-cluster distance measures as well as various validity indexes are defined. These validity indexes include Dunn, Davies Bouldin, Ray-Turi, Xie-Benni validity indexes and the silhouette coefficient. The chapter concludes by listing the measures selected to be used in this research and defining the approach used to determine the quality of a clustering solution in dynamic environments.

Chapter 4

Artificial Datasets

The algorithms implemented in this research have been applied to a variety of datasets. Auto-generated artificial datasets were used to analyse and compare the performance of the algorithms. This chapter describes these artificial datasets.

There is one section in this chapter, other than the summary. The section is as follows:

- Section 4.1 - **Auto-generated Artificial Datasets**: This section summarises the details of the auto-generated datasets used to determine the ability of the proposed algorithms to cluster temporal data.

4.1 Auto-generated Artificial Datasets

Artificial non-stationary datasets are used in this research in order to analyse the clustering ability of the algorithms implemented over a variety of problem types. The artificial non-stationary data, provided by Graaff [46], covers problems of various data migration types, dimensions, severities of change and frequencies of change.

Three data migration types are covered by the datasets, namely pattern migration, cluster migration and centroid migration [46]. A data migration type refers to the manner

in which the data changes. For pattern migration a single data pattern migrates from the cluster to which it belongs to a randomly selected cluster, expanding or compacting clusters. For cluster migration all data patterns in a selected cluster migrate to other randomly selected clusters, resulting in the disappearance of the cluster selected for migration. For centroid migration all data patterns of a cluster migrate to new positions where the patterns still belong to the same cluster.

The severity of change of a dataset refers to the magnitude of the modification that occurs at some point in time [31]. This research uses datasets with severities of 1 to 5 [46], where a higher severity means a larger change.

The frequency of change of the dataset refers to the intervals at which data changes within a fixed period of time [46]. This research uses frequencies of 1 to 5 [46], where a smaller frequency value means that more changes occur within the dataset, and

$$t = \frac{f}{10} * T \tag{4.1}$$

where t is the time-step when a change occurs, T is the total number of time-steps and f is the frequency. To determine the iteration during which a change will occur, equation (4.1) is used, replacing T with iterations.

The dimension of the dataset refers to the total number of attributes of the dataset. This research uses datasets with dimensions of 3, 8 and 15.

A cluster in Graaff's [46] artificial datasets was generated using the multidimensional Gaussian function

$$g(\mathbf{x}_k, \mathbf{c}_k) = a \exp^{-\left[\sum_{n=1}^{N} \frac{(x_{k,n} - c_{k,n})^2}{2\sigma_{k,n}^2}\right]} \tag{4.2}$$

where a is the amplitude, N is the total dimensions of the data patterns being generated, k is the index of the cluster being generated, $x_{k,n}$ is the offset from centroid $c_{k,n}$ in dimension n, and $\sigma_{k,n}$ is the compactness in dimension n.

Once the Gaussian clusters are generated, the non-stationary environment is simulated by moving centroids and patterns along certain paths, depending on the migration type. For centroid migration problems, the paths were generated using an $(N-1)$-dimensional sphere with a radius φ and a middle point $\mathbf{m}$. A centroid $\mathbf{c}_k$ is represented by an angle vector θ_k which is projected onto the surface of the sphere. Changing $\theta_{k,n-1}$ would, therefore, move the centroid $\mathbf{c}_k$ along the surface of the sphere. Patterns are then moved by adding the difference between the previous centroid position and the new centroid position.

The severity of change is used to determine the angle of θ_k by serving as a ratio, $\frac{s}{10}$, of the angle difference between $\theta_{k,n-1}$ and a randomly generated angle sample between $U[0,\pi]$. The frequency of change is also used as a ratio, $\frac{f}{10}$, which is multiplied by the total number of time-steps to determine the time-step at which the change will occur.

Pattern migration and cluster migration problems are simulated by adding and removing patterns to and from various clusters, keeping the total number of patterns per cluster the same.

Table 4.1 summarizes the characteristics of the artificial non-stationary datasets, which result in a total of two hundred and twenty five datasets that were used.

4.2 Summary

This chapter summarised the artificially generated datasets used in this research.

Table 4.1: Characteristics of Artificial Non-stationary Datasets

Parameter	**Value**
Type of data	Numeric
Total data patterns	8000
Total time-steps	100
Total data patterns per time-step	80
Maximum total clusters (K)	8
Migration types	Pattern migration, cluster migration, centroid migration
Dimensions	$\{3, 8, 15\}$
Frequencies	$\{1, 2, 3, 4, 5\}$
Severities	$\{1, 2, 3, 4, 5\}$
a	1
σ	1
φ	15
$\mathbf{m}$	0

Chapter 5

Particle Swarm Optimization Algorithm for Temporal Data Clustering

The work presented in this chapter has been published in [44]. The chapter describes the PSO algorithm and five adaptations made to it in order to cluster temporal data. A variety of PSO algorithms are described, followed by the changes made to the algorithms in order to cluster temporal data. There are three sections in this chapter, each dealing with a separate topic. The sections are organised as follows:

- Section 5.1 - **Particle Swarm Optimisation**: This section summarises the PSO algorithm.
- Section 5.2 - **Data Clustering Particle Swarm Optimisation**: This section introduces the data clustering particle swarm optimisation (DCPSO) algorithm.
- Section 5.3 - **Dynamic Data Clustering Particle Swarm Optimisation**: This section summarises a variety of existing PSOs previously applied to dynamic environments. The chapter then introduces four algorithms implemented to cluster temporal data by combining the DCPSO with the summarised dynamic PSOs.

These algorithms include the reinitialising data clustering particle swarm optimisation (RDCPSO), multi-swarm data clustering particle swarm optimisation (MD-CPSO), cooperative data clustering particle swarm optimisation (CDCPSO) and the cooperative multi-swarm data clustering particle swarm optimisation (CMD-CPSO).

5.1 Particle Swarm Optimization

The particle swarm optimization algorithm [66] is an algorithm that models the social scattering and re-organizing behaviour of birds. It begins by randomly initialising a swarm of particles, which represents a flock of birds, within the search space of the problem. These particles move around in the search space by gaining velocity, which is calculated using:

- the best position found so far by the particle,
- the overall best position found so far by the particle's neighbourhood,
- the previous velocity value,
- two independent random components, and
- two constants that determine the social and personal influence on the particle's velocity. The social influence refers to the influence of the best position found by the particle's neighbourhood, while the personal influence refers to the influence of the best position found by the particle.

A particle is represented by an n-dimensional floating-point vector, where n is the dimensionality of the problem. Each particle represents a possible solution to the optimisation problem, which is a position in the search space, and each particle is updated to find a better position than its current position. The velocity of a particle is also represented as a vector in order to allow for the possibility of each dimension to be updated using a different velocity value. Lastly, the best position found by a particle is stored as the

"personal best" (pbest) position. This personal best position is replaced when a better position is found, and influences the velocity by causing it slowly to pull the particle towards this better position as well as towards the best overall position found by the particle's neighbourhood (the neighbourhood best position).

The quality of a position is determined by a measure called "fitness". Depending on the problem, the fitness measure is calculated differently. For example, for function optimization, the fitness is calculated using the function being optimized. For data clustering, fitness can be calculated as the quantization error, described in Section 3.4.

PSO algorithms form networks, called topologies, which consist of connections between particles in a population [101]. Two popular examples of topologies that have been developed for PSO algorithms are the star topology (used in the global best (gbest) PSO), illustrated in Figure 5.1, and the ring or circle topology (used in the local best (lbest) PSO), illustrated in Figure 5.2 [64]. For the star topology, the best position found by the entire swarm as well as the pbest position of the particle influence the velocity of all particles. For the local best topology a neighbourhood is defined for each particle. The neighbourhood forms a communication structure where a group of selected particles can exchange information about the best positions found so far [4]. The best position in a neighbourhood influences the velocity updates of the particles in the neighbourhood along with those particle's pbest positions.

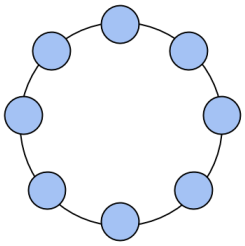


Figure 5.1: Ring topology

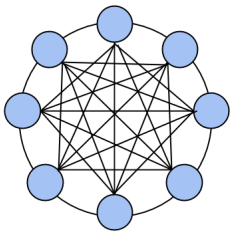


Figure 5.2: Star topology

The original gbest PSO updates velocity using [66]

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)] \tag{5.1}$$

where $v_{ij}(t+1)$ is the velocity value for particle i at dimension j at time step $t+1$, $v_{ij}(t)$ is the velocity value for the same dimension for the previous time step, c_1 is the cognitive acceleration coefficient, c_2 is the social acceleration coefficient, $y_{ij}(t)$ is the personal best position of particle i for dimension j, $x_{ij}(t)$ is the current position of particle i for dimension j, $\hat{y}_j$ is the global best particle's position for dimension j and r_{1j} and r_{2j} are two different random values sampled for each dimension from a uniform distribution, $U(0,1)$.

The cognitive acceleration coefficient determines the influence of the personal best position on the velocity calculation, drawing the new position towards the personal best position [37]. The social acceleration coefficient determines the influence of the neighbourhood or global best position on the velocity calculation, drawing the new position towards the neighbourhood best position [37].

Shi and Eberhart [104] added a parameter called the "inertia weight" in order to control the exploration and exploitation capabilities of the PSO. This parameter determines the influence of the previous velocity of a particle on the new velocity, significantly improving the performance of PSOs. The new velocity calculation is

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)] \tag{5.2}$$

where w is the inertia weight.

The local best PSO calculates velocity by replacing the global best position $\hat{\mathbf{y}}$ with the neighbourhood best position $\hat{\mathbf{y}}_i$ for that particle in equation (5.2).

Other approaches to calculate velocity have been developed by researchers from around the world. Examples of these include velocity clamping [68], constriction velocity calculation [34], cognition only model [63], social only model [63], selfless model [63], fully informed PSO velocity calculation [67] and the barebones velocity calculation [65], amongst others.

Particle positions are calculated as [37]

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{5.3}$$

The standard PSO algorithm using the global best topology is summarised in Algorithm 1, while the local best PSO is summarised in Algorithm 2.

Algorithm 1 Global Best Synchronous PSO Assuming Minimization

Initialise a swarm of N n-dimensional particles as S
while stopping condition has not been reached **do**
    for each particle $\mathbf{x}_i$ in S **do**
        if fitness($\mathbf{x}_i$) < fitness($\mathbf{y}_i$) **then**
            $\mathbf{y}_i = \mathbf{x}_i$
        end if
        if fitness($\mathbf{y}_i$) < fitness($\hat{\mathbf{y}}$) **then**
            $\hat{\mathbf{y}} = \mathbf{y}_i$
        end if
    end for
    for each particle $\mathbf{x}_i$ in S **do**
        Update velocity using equation (5.2)
        Update position using equation (5.3)
    end for
end while

5.2 Data Clustering Particle Swarm Optimization

Data clustering refers to the grouping of similar data patterns in order to determine the relationships among them. A solution to a data clustering problem is a group of centroid positions, where a centroid is the central point of a cluster [118], in other words, the position located in the middle of a group of data patterns which are similar to one another. The aim of the DCPSO is to find an optimal combination of centroid positions for a particular dataset.

Algorithm 2 Local Best Synchronous PSO Assuming Minimization

Initialise a swarm of N n-dimensional particles as S
while stopping condition has not been reached **do**
    for each particle $\mathbf{x}_i$ in S **do**
        if fitness($\mathbf{x}_i$) < fitness($\mathbf{y}_i$) **then**
            $\mathbf{y}_i = \mathbf{x}_i$
        end if
        for each particle $\mathbf{x}_l$ that has particle $\mathbf{x}_i$ in its neighbourhood **do**
            if fitness($\mathbf{y}_i$) < fitness($\hat{\mathbf{y}}_l$) **then**
                $\hat{\mathbf{y}}_l = \mathbf{y}_i$
            end if
        end for
    end for
    for each particle $\mathbf{x}_i$ in S **do**
        Update velocity using equation (5.2)
        Update position using equation (5.3)
    end for
end while

The DCPSO [118] differs slightly from the standard PSO, which was summarised in Algorithm 1. The first difference is the representation of particles. The standard PSO uses a floating-point vector representation for particle positions, where each floating-point value represents a potential best value for the corresponding dimension. For data clustering, a vector representation is not appropriate as the solution to the clustering problem is not a vector.

A particle in a DCPSO is a container which holds a group of centroid positions, as illustrated in Figure 5.3, making the particle an $n * K$-dimensional particle. These centroid positions are updated during each iteration of the algorithm in an attempt to optimally position centroids.

The DCPSO calculates the fitness of a particle according to how well the corresponding

centroids approximate the clusters of the dataset. To calculate particle fitness, each data pattern is assigned to the cluster centroid closest to it. A distance measure, such as the Euclidean distance [37] as defined in equation (3.2), is used to determine the proximity of patterns to centroids. One approach to quantify fitness is to then calculate the quantization error [118] of a particle, defined in equation (3.15), using the assignment of data patterns to centroids.

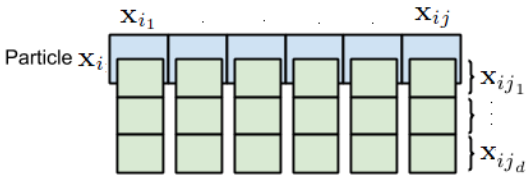


Figure 5.3: Particle representation of a three dimensional clustering problem for six clusters

The rest of the DCPSO algorithm is the same as the standard global best PSO algorithm, as illustrated in Algorithm 3.

5.3 Dynamic Data Clustering Particle Swarm Optimization

The design of the DCPSO poses difficulties in dynamic environments. These difficulties, as discussed in Chapter 2, are the loss of diversity and outdated memory problems. The loss of diversity is caused by the PSO converging around a solution and having no means of re-diversifying the swarm once a change has occurred. The outdated memory problem refers to the fact that, once the data has changed, previous personal best positions (in other words, previous best centroids) held by particles, as well as the neighbourhood best positions, are no longer optimal, yet they may influence the particles to move towards those no longer optimal positions.

A number of existing PSOs previously used for non-stationary environments were adapted in this research to cluster non-stationary data. These adaptations include the RDCPSO,

Algorithm 3 Standard Data Clustering PSO

```
Initialise a swarm, S, of N n*K-dimensional particles
while stopping condition is not reached do
    for each particle x_i in the swarm do
        for each data pattern z_p do
            Calculate the distance between z_p and all centroids
            Assign z_p to the centroid that resulted in the smallest distance
        end for
        Calculate the particle's fitness using equation (3.10)
        Update the global and personal best positions
    end for
    for each particle x_i in the swarm do
        Update velocity using equation (5.2)
        Update position using equation (5.3)
    end for
end while
```

the MDCPSO, the CDCPSO and an adaptation of the CDCPSO which merges the algorithm with the MDCPSO. These algorithms are discussed in the sections that follow.

5.3.1 Reinitialising Data Clustering Particle Swarm Optimization

A reinitialising PSO [36] is a standard PSO that is adapted to re-initialise a percentage of the swarm in order to increase diversity when a change in the environment occurs. This means that a portion of the swarm is placed at random positions around the search space, personal best positions are set to this new position and velocity values are set to zero. The percentage of the swarm that is re-initialised is a problem-specific parameter based on how much diversity has to be injected into the swarm when a change occurs. This algorithm is given in Algorithm 4.

Algorithm 4 Re-initialising PSO

1: Initialise a swarm of N n-dimensional particles
2: **while** stopping condition is not reached **do**
3: Perform an iteration of Algorithm 1
4: **if** a change occurs **then**
5: reinitialise percentage of the swarm
6: **end if**
7: **end while**

The re-initialising PSO is easily adapted to perform data clustering by using the same particle representation as the DCPSO shown in Figure 5.3. Then, during the fitness calculation, the data patterns are assigned to the centroids closest to them. The RDCPSO is the same as Algorithm 4, where an iteration of Algorithm 3 is performed rather than an iteration of Algorithm 1.

One problem with this approach is the loss of previously gained information about the environment caused by the re-initialisation of particles as personal best positions are set to the new random positions of the particles. In an environment with a small severity of change, re-initialising a high portion of the swarm may result in this information loss. For environments with a high severity of change, however, more particles would need to be re-initialised as more of the previously gained information becomes irrelevant. Additionally, re-initialisation of the entire swarm will effectively mean that the algorithm starts the optimization process from the beginning with every change.

Determining how much re-initialisation is required in order to not lose important information as well as re-diversify the swarm enough is a challenge. It is inefficient to re-initialise the entire swarm for every change, yet re-diversification is required for a dynamic environment.

5.3.2 Multi-swarm Data Clustering Particle Swarm Optimisation

Blackwell and Branke [10] adapted the standard PSO in order to overcome the problems it faced in dynamic environments. The proposed algorithm is the multi-swarm particle swarm optimisation (MPSO), where a number of swarms are generated, each of which searches for one optimum at a time. Two mechanisms allow for the algorithm to adapt to dynamic environments, namely repulsion and anti-convergence. Repulsion introduces diversity ensuring that two sub-swarms will not exploit the same solution. Sub-swarms are repelled by re-initialising the weakest of the sub-swarms if two sub-swarm's global best positions are within a certain distance from each other, determined by an exclusion radius. Here the weakest sub-swarm refers to the sub-swarm whose global best position has the worst fitness value between the two sub-swarms. Anti-convergence introduces further diversity by re-initialising the weakest sub-swarm when all sub-swarms have converged. This allows the algorithm to continuously explore the search space for new solutions, requiring no mechanism to track whether a change in the environment has occurred. Blackwell and Branke's MPSO is summarised in Algorithm 5.

For data clustering, each particle already holds all the centroids and all centroids are required in order to calculate the fitness of the particle. For this reason, the original multi-swarm algorithm, where each sub-swarm searches for one of multiple solutions, was adapted. The new algorithm has a group of sub-swarms where each swarm optimises all centroids instead of one centroid. The representation of the particle is then the same as the representation used by the DCPSO, illustrated in Figure 5.3. The MDCPSO is therefore effectively an ensemble of DCPSO algorithms.

The MDCPSO uses the same repulsion and anti-convergence mechanisms as Blackwell and Branke's MPSO. Fitness is calculated by assigning each data pattern to the centroid closest to it and the final solution found by the algorithm is the best global best position over all sub-swarms. The MDCPSO algorithm is the same as Algorithm 5, where an iteration of Algorithm 3 is performed instead of an iteration of Algorithm 1.

Algorithm 5 Multi-swarm PSO

Initialise s swarms $S_k, k = 1, ..., s$
Initialise r_{excl}, the exclusion radius
Initialise r_{conv}, the convergence radius
while stopping condition has not been reached **do**
    if all swarms have a diameter which is smaller than r_{conv} **then**
        Reinitialise the worst one by placing its individuals in random positions of the search space
    end if
    for each swarm S_k **do**
        Perform an iteration of algorithm Algorithm 1 for S_k
    end for
    for each swarm S_k **do**
        for each other swarm S_l **do**
            if distance between $\hat{\mathbf{y}}_k$ and $\hat{\mathbf{y}}_l < r_{excl}$ **then**
                if fitness($\hat{\mathbf{y}}_k$) < fitness($\hat{\mathbf{y}}_l$) **then**
                    Reinitialise S_k by placing its individuals in random positions of the search space
                else
                    Reinitialise S_j by placing its individuals in random positions of the search space
                end if
            end if
        end for
    end for
end while

5.3.3 Cooperative Data Clustering Particle Swarm Optimization

The cooperative particle swarm optimisation (CPSO) algorithm, given in Algorithm 6, was created by Van den Bergh and Engelbrecht [117]. It is a multi-swarm approach, different from the MPSO of Blackwell and Branke [10], where an n-dimensional problem

is solved by creating n 1-dimensional sub-swarms. Each swarm individually optimizes one dimension of the problem. A particle representing the complete solution to the problem is created by combining the best position from each sub-swarm. This particle is called the context particle, and is used to calculate the fitness of each solution.

To calculate the fitness of a solution belonging to sub-population S_k, the dimension k of the context particle is replaced with that solution. The remainder of the context particle consists of the best positions of the other populations. The fitness of this adapted context particle is then calculated and assigned as the fitness of the solution. Figure 5.4 shows an example of how the context particle is used to build a particle on which to calculate fitness. In the figure, the context particle consists of centroids A, B, C, D and E and the particle's current position is centroid F. The particle, p_{3_1}, belongs to swarm S_3 and, therefore, replaces the third dimension of the context particle, centroid C. The fitness of particle p_{3_1} is then calculated on a particle consisting of centroids A, B, F, D and E.

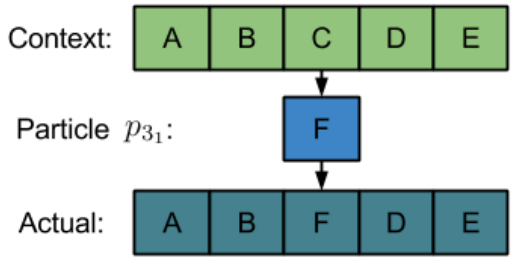


Figure 5.4: Cooperative PSO Fitness Calculation

The CPSO can be adapted to perform the data clustering task by using a swarm of K sub-swarms, where each sub-swarm S_k optimises one centroid at a time. The CPSO allows for different populations to optimise different centroids by using the context particle, giving the clustering problem information about all the centroids in order to calculate fitness. The context particle is then a group of centroids and the data patterns are assigned to their closest centroids prior to the fitness calculation. The CDCPSO is the same as Algorithm 6, where each pattern is assigned to the centroid closest to it before the fitness is calculated.

A combination of a CPSO and a MPSO was implemented, referred to as the CMDCPSO.

Algorithm 6 Cooperative PSO

Initialise K swarms of one-dimensional particles
Calculate initial fitness of each particle

Generate a context individual, and denote as $\mathbf{b}(k, S_k.\hat{\mathbf{y}}_i)$, where k is the population index and $S_k.\hat{\mathbf{y}}_i$ is the best position for that population.
while stopping condition has not been reached **do**
    for each sub-swarm S_k **do**
        for each particle $\mathbf{x}_i$ in S_k **do**
            if fitness$(\mathbf{b}(k, S_k.\mathbf{x}_i) <$ fitness$(\mathbf{b}(k, S_k.\mathbf{y}_i))$ **then**
                $S_k.\mathbf{y}_i = S_k.\mathbf{x}_i$
            end if
            if fitness$(\mathbf{b}(k, S_k.\mathbf{y}_i) <$ fitness$(\mathbf{b}(k, S_k.\hat{\mathbf{y}}_i))$ **then**
                $S_k.\hat{\mathbf{y}}_i = S_k.\mathbf{y}_i$
            end if
        end for
        for each particle $\mathbf{x}_i$ in S_k **do**
            Update velocity using equation (5.2)
            Update position using equation (5.3)
            Calculate fitness of $\mathbf{x}_i$ using $S_k.\mathbf{x}_i$
        end for
        Select survivors for next generation
    end for
end while

The CMDCPSO adapts the DCPSO based on Blackwell and Branke's [10] multi-swarm approach so that each sub-swarm optimizes one centroid instead of all centroids. Figure 5.5 shows an example of a solution in the CMDCPSO and Figure 5.6 shows an example of a solution in a MDCPSO. In both figures the best position is surrounded by a rectangle labelled "solution". Figure 5.5 shows three swarms' 1-dimensional best positions and the context particle created from these, which is the solution. Figure 5.6

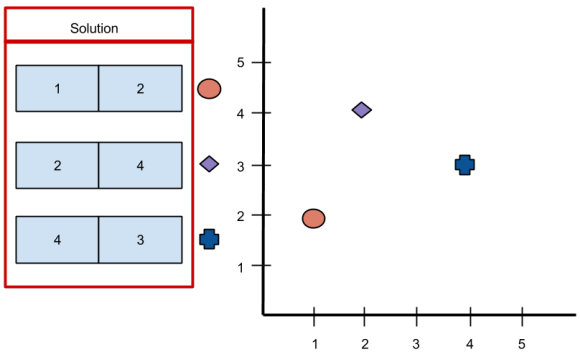


Figure 5.5: Example of Centroid Positions of Global Best Particles From Three Swarms in a Cooperative Multi-swarm PSO

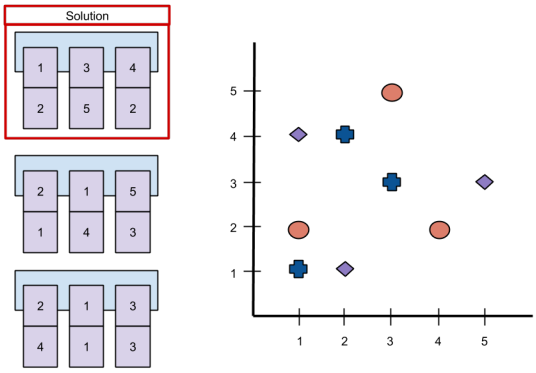


Figure 5.6: Example of Centroid Positions of Global Best Particles From Three Swarms in a Multi-swarm PSO

illustrates the same scenario, where the multi-swarm algorithm optimises complete solutions in parallel, and therefore each swarm's best position consists of all three centroids, making the best position be the global best position over all swarms.

The CDCPSO uses a context particle like the one used in the CDCPSO to calculate fitness. The difference between the CMDCPSO and CDCPSO is the introduction of repulsion and anti-convergence mechanisms borrowed from Blackwell and Branke's MPSO. These mechanisms deal with the main two problems that PSOs experience in dynamic environments, namely the loss of diversity and outdated memory. By using repulsion and anti-convergence the swarms are re-diversified and memory of the particles is updated due to the re-initialisation of particles, personal best positions and velocities.

Because the context particle consists of the global best position of every sub-swarm, the anti-convergence mechanism, where the weakest swarm is re-initialised if all swarms are converging, causes one of the centroids of the final solution to be a re-initialised centroid. This re-initialisation makes the context particle an inaccurate representation of the solution as one of the swarms will always be exploring the search space. To remedy this, the algorithm initialises K + 1 swarms, where the extra swarm is called an "explorer swarm". When a swarm is re-initialised by anti-convergence it is given the status of "explorer" and its solution is not used in the creation of the context particle.

Once a new explorer is assigned, the previous explorer takes that swarm's place. This ensures that the context particle only consists of solutions taken from converging swarms.

The algorithm, summarised in Algorithm 7, begins by initialising a context particle and the swarms. It then updates the context particle, calculates the fitness of each particle, and performs an iteration of the MDCPSO.

The CMDCPSO changes the context particle to include the best position of each swarm regardless of whether the fitness of the context particle itself will be better than that of the previous context particle. An elitist version of the algorithm was developed in order to evaluate the effect of only changing the context particle if the new context particle's value is better than the old one. This elitist version only changes the context particle to include the new best position if the new context particle will have a better fitness than the old one, introducing additional exploitation.

5.4 Summary

This chapter described the standard PSO and DCPSO algorithms. The problems faced by the DCPSO when clustering non-stationary data were discussed and the algorithms developed to overcome those downfalls were introduced.

Algorithm 7 Cooperative-Multiswarm PSO

Initialise $K+1$ swarms of n dimensional particles, where n is the dimension of a data pattern
for each sub-swarm S_k **do**
    initialise particles
end for
Generate a context individual, and denote as $b(k, S_k.\hat{\mathbf{y}}_i)$, where k is the population index and $S_k.\hat{\mathbf{y}}_i$ is the best position for that population. sub-swarm
while stopping condition is not reached **do**
    for each sub-swarm S_k in swarm P **do**
        for each particle $\mathbf{p}_i$ in S_k **do**
            Replace dimension k of $b(k, S_k.\hat{\mathbf{y}}_i)$ with $S_k.\mathbf{x}_i$ to create $b(k, S_k.\mathbf{x}_i)$
            for each data pattern $\mathbf{z}_p$ **do**
                Calculate the Euclidean $d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j})$ of $\mathbf{z}_p$ and each cluster centroid $S_k.\mathbf{x}_{ik_j}$ of $b(k, S_k.\mathbf{x}_i)$
                Assign pattern $\mathbf{z}_p$ to centroid $S_k.\mathbf{x}_{ik_j}$, such that $d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j}) = min_{\forall k_j=1\ldots K_j}\left\{d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j})\right\}$
            end for
            Calculate the fitness of $b(k, S_k.\mathbf{x}_i)$
        end for
        for each particle $\mathbf{x}_i$ in S_k **do**
            if fitness$(\mathbf{b}(k, S_k.\mathbf{x}_i) <$ fitness$(\mathbf{b}(k, S_k.\mathbf{y}_i))$ **then**
                $S_k.\mathbf{y}_i = S_k.\mathbf{x}_i$
            end if
            if fitness$(\mathbf{b}(k, S_k.\mathbf{y}_i) <$ fitness$(\mathbf{b}(k, S_k.\hat{\mathbf{y}}_i))$ **then**
                $S_k.\hat{\mathbf{y}}_i = S_k.\mathbf{y}_i$
            end if
        end for
    end for
    Perform one Multi-swarm iteration, Algorithm 5 using the context particle
end while

Chapter 6

Differential Evolution Algorithm for Temporal Data Clustering

The work presented in this chapter has been published in [45]. The chapter describes the DE algorithm and three adaptations made to it in order to cluster temporal data. Three DE algorithms are described, followed by the changes made to these algorithms in order to cluster temporal data. There are three sections in this chapter, each dealing with a separate topic. The sections are organised as follows:

- Section 6.1 - **Differential Evolution**: This section summarises the DE algorithm.
- Section 6.2 - **Data Clustering Differential Evolution**: This section introduces the data clustering differential evolution (DCDE) algorithm.
- Section 6.3 - **Dynamic Data Clustering Differential Evolution**: This section summarises a variety of existing DEs previously applied to dynamic environments. The chapter then introduces three algorithms implemented to cluster temporal data by combining the DCDE with the summarised dynamic DEs. These algorithms include the re-initialising data clustering differential evolution (RDCDE), dynamic data clustering differential evolution (DCDynDE), and the CDCDynDE.

6.1 Differential Evolution

DE algorithms [109] are population-based evolutionary algorithms guided by distance information about individuals in the population. The population consists of vectors, called individuals, which are adapted via the three evolutionary processes of mutation, crossover and selection. The algorithm begins by randomly initialising the population of individuals to positions within the search space. An individual is mutated by creating a new individual, referred to as the trial individual, using the difference between two or more randomly selected individuals. An offspring is then created by discrete recombination of the new individual and the parent. Lastly, the survivor for the next iteration is selected between the resulting offspring and the original individual based on a comparison of both the offspring and parent fitness.

For each parent individual, an offspring is generated through application of mutation and crossover. During the mutation process, an individual, called a target individual, is selected from the population. This target individual can be selected at random, the individual from the population with the best position, or a linear combination of the individual with the best position and a randomly selected one. Additionally, if only one difference individual is used, two random individuals, different from the target and parent individuals, are selected and the difference between them is calculated. This difference individual gives the mutation process distance information about the individuals in population. The difference is then scaled and added to the target individual, creating a trial individual using

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)) \tag{6.1}$$

where $\mathbf{x}_{i_1}(t)$ is the target individual, β is the scaling factor, and $\mathbf{x}_{i_2}(t)$ and $\mathbf{x}_{i_3}(t)$ are randomly selected individuals from the population, where $i_1 \neq i_2 \neq i_3 \neq i$.

Alternative strategies for mutating individuals have been explored [25][37][91][97][108]. These alternatives may involve using more than one difference individual, as well as using the global best position or position of the current individual of the population to

calculate the difference individuals.

The mutation process does not use information about the parent individual. The parent individual is instead used in the binomial crossover process, which generates one offspring using [37][136]

$$\mathbf{x}'_{ij}(t) = \begin{cases} \mathbf{u}_{ij} & \text{if } j \text{ is} \in J \\ \mathbf{x}_{ij}(t) & \text{if } j \text{ is} \notin J \end{cases} \tag{6.2}$$

where u_{ij} is dimension j of trial individual $\mathbf{u}_i$, x_{ij} is dimension j of the parent individual $\mathbf{x}_i$, and J is a set of crossover points, which are randomly selected. A crossover probability is used to determine which crossover points will be part of J, ensuring that one randomly selected dimension is forced to be within the set J.

Exponential crossover [37][136] is another crossover strategy. In exponential crossover a set of consecutive points starting from a randomly selected point is added to J, instead of a set of randomly selected non-adjacent points. The set is of a random size and the individual is treated as a circular structure, where the dimension following the last dimension of the individual is the first dimension of the individual, as illustrated in Figure 6.1.

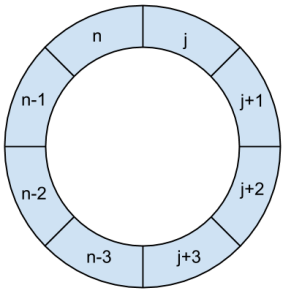


Figure 6.1: Circular Representation of an Individual for Exponential Crossover

Selection is the last process performed when updating an individual of the population. The process involves choosing an individual to survive to the next iteration. The original DE [109] uses elitist selection, where the individual with the best fitness value survives

to the next iteration. The influence of various other selection strategies, however, has been previously analysed [77].

The DE algorithm using a single difference vector is summarised in Algorithm 8.

Algorithm 8 Differential Evolution

```
Initialise a population S of N individuals
while stopping condition has not been reached do
    for each individual x_i in S do
        Calculate the fitness of the individual, x_i
        Select a target individual and two difference individuals
        Generate a individual entity u_i(t) using equation (6.1)
        Generate the offspring x'_i using equation (6.2)
        Calculate the fitness of the offspring, x'_i
        if f(x'_i) is better than f(x_i) then
            add x'_i to the next population
        else
            add x_i to the next population
        end if
    end for
end while
```

6.2 Data Clustering Differential Evolution

The purpose of the DCDE is the same as the purpose of the data clustering PSO. The algorithm aims to find an optimal combination of centroid positions for a particular dataset in order to infer relationships among data patterns.

The DCDE [51] differs in two aspects from the standard DE, in the same way that the DCPSO differs from the standard PSO. The first difference is the representation of the individuals, where an individual represents a solution as a group of centroids. The second difference is the calculation of the fitness, which relies on data patterns being assigned

to the centroids closest to the data pattern. The DCDE is summarised in Algorithm 9.

Algorithm 9 Data Clustering DE

Initialise a population S of $n * K$-individuals
while stopping condition has not been reached **do**
  for each individual x_i in S **do**
    for each data pattern $\mathbf{z}_p$ **do**
      Calculate the Euclidean distance, $d(\mathbf{z}_p, \mathbf{x}_{ij})$, to all cluster centroids j
      Assign pattern $\mathbf{z}_p$ to cluster C_{ij}, such that $d(\mathbf{z}_p, \mathbf{x}_{ij})$ $= min_{\forall k=1...K} \{d(\mathbf{z}_p, \mathbf{x}_{ik})\}$
    end for
    Calculate the fitness of x_i
    Select the target and difference individuals
    Generate a trial individual $u_i(t)$ using equation (6.1)
    Generate the offspring x'_i using equation (6.2)
    if $f(x'_i)$ is better than $f(x_i)$ **then**
      add x'_i to the next population
    else
      add x_i to the next population
    end if
  end for
end while

6.3 Dynamic Data Clustering Differential Evolution

The DCDE [51] faces a problem in dynamic environments: it suffers from loss of diversity in the population. As the algorithm performs the search, the population begins to converge around the best found position. This convergence means that all individuals in the population begin to move towards the same position, leaving no individuals to explore the search space for changes if any occur. When a change in the dataset does occur, the individuals may already be exploiting the best position and may therefore not

adapt to the change.

This section describes three adaptations of the standard DE that attempt to address the problem of diversity loss as well as the algorithms' data clustering alternatives. The modified DEs are the re-initialising DE, re-initialising DCDE, DynDE, DCDynDE and the CDCDynDE.

6.3.1 Re-initialising Differential Evolution

The re-initialising differential evolution (RDE) algorithm is implemented in this research to adapt the standard DE by re-initialising a percentage of the population when a change in the environment is detected, where the percentage of individuals to be re-initialised is a problem specific parameter. The re-initialisation places individuals at random positions within the bounds of the search space, injecting diversity into the population. The RDE is summarised in Algorithm 10.

Algorithm 10 Re-initialising DE

```
Initialise a population S of N K-dimensional individuals
while stopping condition is not reached do
    Perform an iteration of Algorithm 8
    if a change occurs then
        Reinitialise a percentage of the population
    end if
end while
```

The RDE is easily adapted to perform data clustering by using the same individual representation as DCDE. The is the same as Algorithm 10, but performs an iteration of Algorithm 9 instead of Algorithm 8. The algorithm faces the same problems as the re-initialising PSO as described in Section 5.3.1.

6.3.2 Dynamic Differential Evolution

The DynDE [80] is a dynamic DE that adapts the standard DE by adding two additional steps and converting it to a multi-population algorithm. The DE algorithm was restructured by Mendes and Mohais [80] to a multi-population algorithm with s sub-populations where each sub-population searches for a solution.

The first additional step of the DynDE algorithm is exclusion. If two sub-populations' global best positions are within an exclusion radius r_{excl} of each other they are repelled, leaving one sub-population intact and re-initialising all the individuals of the sub-population with the worst global best fitness to random positions within the search space. This allows for populations to explore different areas of the search space and inherently adapt to the changing dataset.

The second change is the addition of Brownian individuals to the sub-population. Brownian individuals are not adapted using the standard mutation and crossover operators. Instead, their positions are adapted based on the global best position within the sub-population instead. Each sub-population has a percentage of Brownian individuals whose position is adapted by adding Gaussian noise to the global best individual's position using

$$w(x_i j) = y_j + N(0, \sigma) \tag{6.3}$$

where $w(x_i j)$ is the new value for dimension j of the Brownian individual, y_j the value for dimension j of the global best individual's position, and $N(0, \sigma)$ is Gaussian noise with a mean of 0 and a standard deviation of σ. The σ value defines the width of the Gaussian distribution, making a large σ value promote exploration and a small σ value promote exploitation. During every iteration, a set percentage of the weakest individuals is selected to become Brownian individuals.

Mendes and Mohais [80] also evaluated the effect of quantum individuals and entropic DE on the DynDE algorithm instead of using Brownian individuals. The study concluded

that the best results were achieved using Brownian individuals.

The DynDE algorithm begins by initialising the sub-population's individuals to random positions within the search space. The fitness of each individual in each sub-population is then calculated. The algorithm continues by calculating the distance between the global best positions of each sub-population, and, if two population's global best positions are within r_{excl} of each other, the weakest sub-population is re-initialised. The sub-populations that are not re-initialised update their individuals using the mutation, crossover and selection operators of the DCDE. A percentage of the weakest individuals in the sub-population is chosen to be Brownian individuals and each Brownian individual is updated using equation (6.3). The DynDE algorithm is summarised in Algorithm 11.

Algorithm 11 Dynamic DE

1: Initialise s sub-populations of individuals
2: **while** stopping condition is not reached **do**
3: Evaluate each sub-population by performing the steps described in lines 4-8 of Algorithm 9
4: Compare the global best individual position from each sub-population to each other
5: **if** the global best positions of two sub-populations are within r_{excl} of each other **then**
6: Re-initialise the sub-population with the worst global best position
7: **else**
8: **for** each sub-population S_k that was not re-initialised **do**
9: **for** each individual $\mathbf{x}_i$ in S_k **do**
10: Update $\mathbf{x}_i$ using lines 10-12 of Algorithm 8
11: **end for**
12: **end for**
13: Make a percentage of the weakest individuals Brownian individuals and update them using equation (6.3)
14: **end if**
15: **end while**

The DynDE algorithm is adapted for the data clustering problem by changing the representation of a solution to be a group of centroids instead of a vector. Each sub-population optimises all the centroids and the final solution is the best global best position over all sub-populations. The algorithm remains the same as Algorithm 11 where an iteration of Algorithm 9 is performed instead of an iteration of Algorithm 8. Brownian individuals are updated by adding Gaussian noise to each dimension of each centroid of the global best individual's position. Lastly, fitness is calculated by first assigning data patterns to the data pattern's closest centroids.

6.3.3 Cooperative Data Clustering Dynamic Differential Evolution

The DCDynDE algorithm consists of sub-populations where each sub-population searches for all centroid positions of a data clustering problem. Multi-population algorithms, however, allow for each sub-population to search for one solution, rather than for all solutions. In the case of the DCDynDE, sub-populations could not be separated in a way that each sub-population searches for one centroid due to the fact that the fitness function requires information about all centroids. The CDCDynDE aims to solve this problem by introducing a context individual, inspired by the cooperative PSO developed by Van den Bergh and Engelbrecht [117]. This context individual is used to determine the fitness of an individual in the same manner as the CDCPSO, MDCPSO, and CMDCPSO algorithms described in Section 5.3.3.

The rest of the data clustering cooperative DynDE algorithm behaves in the same way as the DCDynDE. The sub-populations are initialised, fitness is calculated using the context individual, sub-populations optimising the same area are repelled and sub-populations are updated using DE operators or the Brownian individual update in equation (6.3). The CDCDynDE algorithm is summarised in Algorithm 12.

Algorithm 12 Cooperative Data Clustering Dynamic DE

Initialise s sub-populations of individuals
while stopping condition is not reached **do**
    Generate a context individual, and denote as $b(k, S_k.\hat{\mathbf{y}}_i)$, where k is the population index and $S_k.\hat{\mathbf{y}}_i$ is the best position for that population.
    for each sub-population S_k **do**
        for each individual $S_k.\mathbf{x}_i$ **do**
            Replace dimension k of $b(k, S_k.\hat{\mathbf{y}}_i)$ with $S_k.\mathbf{x}_i$ to create $b(k, S_k.\mathbf{x}_i)$
            for each data pattern $\mathbf{z}_p$ **do**
                Calculate the Euclidean $d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j})$ of $\mathbf{z}_p$ and each cluster centroid $S_k.\mathbf{x}_{ik_j}$ of $b(k, S_k.\mathbf{x}_i)$
                Assign pattern $\mathbf{z}_p$ to centroid $S_k.\mathbf{x}_{ik_j}$, such that $d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j}) = min_{\forall k_j=1...K_j}\left\{d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j})\right\}$
            end for
            Calculate the fitness of $b(k, S_k.\mathbf{x}_i)$
        end for
    end for
    Compare the global best positions from each sub-population to each other
    if the global best positions of both sub-populations are within r_{excl} of each other **then**
        re-initialise the sub-population with the worst global best position
    else
        for each sub-population that was not re-initialised **do**
            for each individual $S_k.\mathbf{x}_i$ **do**
                Update $S_k.\mathbf{x}_i$ using the mutation, crossover and selection mechanisms of the DE
            end for
            Make a percentage of the weakest individuals Brownian Individuals and update them using Equation (6.3)
        end for
    end if
end while

6.4 Summary

This chapter described the standard DE and DCDE algorithms. The problems faced by the DCDE when clustering non-stationary data are discussed and algorithms were developed to overcome these problems.

Chapter 7

Local Network Neighbourhood Artificial Immune System Overview

This chapter summarises AIS algorithms. The network AIS model and the LNNAIS, which is an algorithm designed for applications in data clustering, are discussed. There are two sections in this chapter, each dealing with a separate topic. The sections are organised as follows:

- Section 7.1 - **Artificial Immune System**: This section summarises AIS algorithms.
- Section 7.2 - **Local Network Neighbourhood Artificial Immune System for Data Clustering**: This section summarises the LNNAIS algorithm for data clustering.

7.1 Artificial Immune System

AIS algorithms [38] [46] are population based algorithms inspired by the natural immune system of vertebrates. Like the natural immune system, an AIS trains to recognise un-

known patterns in the system. These unknown patterns, referred to as antigen patterns, become the training set for the algorithm.

The AIS algorithm begins by initialising a population of individuals, named ALCs, to random positions around the search space [37] or as a sub-set of the training data [115]. An ALC is represented by a vector of the same dimensionality as the data in the training set. For each antigen pattern, ALCs are selected from a random subset of the ALC population. The size of the sub-set is a user defined parameter. The ALCs most similar to the antigen pattern are then adapted.

The AIS algorithm calculates the similarity between an antigen pattern and an ALC in order to determine which ALCs to adapt. This similarity measure is referred to as the antigen affinity and is typically calculated using a distance measure, such as the Euclidean distance defined in equation (3.2). As the distance between an antigen pattern and an ALC decreases, the antigen affinity, or similarity, increases. The antigen affinity between an ALC i and an antigen pattern p is therefore calculated as [115]

$$a(\mathbf{z}_p, \mathbf{x}_i) = m - d(\mathbf{z}_p, \mathbf{x}_i) \tag{7.1}$$

where $m = 1$ and $a(i, p)$ is normalised such that $0 < a(i, p) < 1$.

Various AIS models exist, such as classical models, clonal selection models, network models and danger theory models [37]. This research focuses on network models [115], where each ALC is not only stimulated by the antigen pattern, but also by other ALCs. In network models the update of an ALC is influenced by a neighbourhood affinity, neighbourhood suppression, and stimulation level in addition to the antigen affinity. The neighbourhood affinity refers to the affinity between ALCs, indicating the degree to which ALCs are part of the same neighbourhood. This neighbourhood affinity is calculated using

$$na(\mathbf{x}_i) = \sum_{j=0}^{N_i} m - d(\mathbf{x}_i, \mathbf{x}_j) \tag{7.2}$$

where $\mathbf{x}_j$ is an ALC in the neighbourhood of $\mathbf{x}_i$, $m = 1$, N_i is the size of the neighbourhood of $\mathbf{x}_i$ and $na(i, j)$ is normalised such that $0 < na(i, j) < 1$.

In order to calculate the neighbourhood affinity, the neighbourhood of an ALC must be determined. An ALC $\mathbf{x}_i$ is considered the neighbour of an ALC $\mathbf{x}_j$ if the affinity between the two ALCs exceeds a network affinity threshold. The network affinity threshold is calculated using

$$NAT = \frac{A}{N}\left\{\sum_{i=0}^{N}\sum_{j=0}^{N} d(\mathbf{x}_i, \mathbf{x}_j)\right\} \tag{7.3}$$

where $0 < A < 1$ is a user defined constant used to control connectivity.

The neighbourhood suppression refers to the degree of dissimilarity between ALCs, calculated using

$$ns(\mathbf{x}_i) = -\sum_{j=0}^{n} d(\mathbf{x}_i, \mathbf{x}_j) \tag{7.4}$$

where $ns(\mathbf{x}_i)$ is the normalised network suppression such that $0 < ns(\mathbf{x}_i) < 1$.

The stimulation level refers to the degree to which an antigen pattern triggers an immune response from an ALC. The stimulation level of an ALC combines the antigen affinity, neighbourhood affinity and neighbourhood suppression, using

$$sl(\mathbf{z}_p, \mathbf{x}_i) = a(\mathbf{z}_p, \mathbf{x}_i) + na(\mathbf{x}_i) + ns(\mathbf{x}_i) \tag{7.5}$$

where $a(\mathbf{z}_p, \mathbf{x}_i)$ is the antigen affinity between an ALC $\mathbf{x}_i$ and data pattern $\mathbf{z}_p$, $na(\mathbf{x}_i)$ is the network affinity of the ALC and $ns(\mathbf{x}_i)$ is the network suppression of the ALC.

When the stimulation level of an ALC surpasses a user predefined threshold the ALC is cloned. The clonal rate for the ALC is calculated using

$$e_x = sl(i)G \tag{7.6}$$

where G is a user defined constant.

The cloned ALCs are mutated in order for the AIS to adapt to variations of the antigen pattern as well. During the mutation process, each dimension of the ALC has a probability of being mutated. The probability, referred to as the mutation rate, is a user defined parameter. Once a dimension is selected for mutation, the value of the dimension is replaced with a randomly generated value.

The natural immune system disposes of an estimate of 5% of cells daily. The AIS models this process by disposing of 5% of the ALC population according to network affinity values.

The network AIS model is summarised in Algorithm 13.

The AIS algorithm can be used for data clustering without any alterations. The data to be clustered is the collection of antigen patterns given to the ALC as the training set. Each resulting ALC network represents a cluster found by the AIS.

7.2 Local Network Neighbourhood Artificial Immune System for Data Clustering

Graaff [46] proposed the LNNAIS, which is an AIS algorithm for applications in data clustering. The LNNAIS algorithm adapts the network AIS by changing the process of determining an ALC's neighbourhood.

ALCs in the population of a LNNAIS are initialised to random positions within the bounds of the dataset to be clustered. The affinity calculations of the algorithm remain the same as those for the network AIS, namely equation (7.1) and equation (7.2), where $m = 0$.

Algorithm 13 Network Model of AIS

```
Initialise a population of s n-dimensional ALCs as S
Determine the antigen pattern training collection as D_T
while stopping condition has not been reached do
    for each ALC x_i in S do
        for each ALC x_j in S do
            Calculate the affinity between x_i and x_j
        end for
    end for
    Calculate NAT
    for each antigen pattern z_p in D_T do
        for each ALC x_i in S do
            Determine the neighbourhood of x_i using NAT
            Calculate the neighbourhood affinity na(x_i)
            Calculate the neighbourhood suppression ns(x_i)
            Calculate the antigen affinity a(z_p, x_i)
            Calculate the stimulation level sl(x_i)
            if sl(x_i) > THRESHOLD then
                Clone x_i at a clonal rate e_x
                Mutate the clones at a pre-determined mutation rate
            end if
        end for
    end for
end while
```

The ALC $\mathbf{w}_h$ with the highest calculated affinity is selected. The antigen pattern is added to the set of clones Cl_h of $\mathbf{w}_h$ as an antigen mutated clone. The addition of this antigen mutated clone increases the clonal level of the ALC. When the clonal level of the ALC exceeds a threshold, the ALC is activated and creates a mutated ALC clone using

$$\mathbf{w}'_h = \mathbf{w}_h + \frac{\sum_{g=1}^{|Cl_h|} a^*(\mathbf{w}_h, \mathbf{a}'_g, Cl_h)(\mathbf{w}_h - \mathbf{a}'_g)}{\sum_{c=1}^{|Cl_h|} a^*(\mathbf{w}_h, \mathbf{a}'_g, Cl_h)} \tag{7.7}$$

where $\mathbf{a}'_g$ is an antigen mutated clone, $|Cl_h|$ is the size of the clonal set of $\mathbf{w}_h$ and $a*(\mathbf{w}_h, \mathbf{a}'_g, Cl_h)$ is the normalised affinity calculated using

$$a^*(\mathbf{w}_h, \mathbf{a}'_c, Cl_h) = 1 - \frac{a(\mathbf{w}_h, \mathbf{a}'_c)}{a_{max} + 1.0} \tag{7.8}$$

where a_{max} is the maximum affinity between the ALC and the antigen mutated clones in the ALCs clonal set Cl_h.

The mutated ALC clone $\mathbf{w}'_h$ inherits the antigen mutated clones of $\mathbf{w}_h$ with which $\mathbf{w}'_h$ has a higher affinity than $\mathbf{w}_h$. If more than half of the antigen mutated clones of $\mathbf{w}_h$ are added to the clonal set of $\mathbf{w}'_h$, then $\mathbf{w}_h$ is added to the clonal set of $\mathbf{w}'_h$. On the other hand, if less than half of the antigen mutated clones of $\mathbf{w}_h$ are added to the clonal set of $\mathbf{w}'_h$, then $\mathbf{w}_h$ is suppressed by removing all its antigen mutated clones.

The neighbourhood of an ALC in the LNNAIS is determined by a network neighbourhood window of size p and the initial index of the ALC. The neighbourhood of an ALC $\mathbf{x}_i$ consists of all ALCs in the population between indexes $i - (p-1)$ and $i + (p-1)$.

After an ALC is cloned and mutated, the stimulation level of $\mathbf{w}_h$ is adapted. The immediate neighbours of $\mathbf{w}_h$, namely $\mathbf{w}_{h-1}$ and $\mathbf{w}_{h+1}$, react to the $\mathbf{w}_h$ antibody by adding the set of clones of $\mathbf{w}_h$ to their respective set of clones. If either of the immediate neighbours become activated, the activated ALCs will stimulate their immediate neighbours, $\mathbf{w}_{h-2}$ and $\mathbf{w}_{h+2}$ respectively. If a neighbouring ALC is not activated by its neighbour's stimulation, then the stimulating ALC is inserted into the neighbourhood at the index of the ALC that was not activated, referred to as clonal expansion and shifting the population's indices. The process continues until the boundaries of the local network are reached or until a neighbouring ALC is not activated. Lastly, two ALCs with the highest network affinity in the population, excluding ALCs within the local neighbourhood, are

merged in order to decrease the population size which may have increased through clonal expansion.

The LNNAIS algorithm is summarised in Algorithm 14.

Algorithm 14 LNNAIS

Set the maximum size of the ALC population
Initialise an empty set of ALCs as population S
for each antigen pattern $\mathbf{z}_p$ in D_T **do**
    if population P is empty **then**
        Initialise a new ALC $\mathbf{x}_i$ with the same structure as $\mathbf{z}_p$
        Add $\mathbf{x}_i$ to S
    end if
    Calculate the antigen affinity between $\mathbf{z}_p$ and each $\mathbf{x}_i$ in S
    Select the ALC with the highest antigen affinity as $\mathbf{w}_h$
    Add the antigen mutated clone to $\mathbf{w}_h$
    if $\mathbf{w}_h$ is activated **then**
        Generate mutated clone $\mathbf{w}'_h$
        Determine the local network neighbourhood of $\mathbf{w}_h$
        Adapt the local network neighbourhood of $\mathbf{w}_h$
    end if
end for

7.3 Summary

This chapter summarised the AIS algorithm, the network AIS model and the LNNAIS algorithm. The purpose of the chapter is to introduce the reader to the LNNAIS algorithm against which results of this research are compared.

Chapter 8

Dynamic Differential Evolution with Automatic Determination of the Number of Clusters

This chapter introduces the KCDCDynDE algorithm. The algorithm adapts the CDCDynDE, which needs prior knowledge of the number of clusters, to determine the optimal number of clusters dynamically. There is one section in this chapter, apart from the summary of the chapter. The section is as follows:

- Section 8.1 - **K-independent Cooperative Data Clustering Dynamic Differential Evolution**: This section summarises the proposed KCDCDynDE algorithm.

8.1 K-independent Cooperative Data Clustering Dynamic Differential Evolution

The CDCDynDE algorithm requires prior knowledge of the optimal number of clusters to be found. In real-datasets the number of clusters, K, is unknown, requiring any algorithm used to cluster the data to determine K automatically or through parameter tuning. An adaptation of the CDCDynDE is proposed in this section. This adaptation does not need prior knowledge of K and determines the number of clusters dynamically, adapting K to dataset changes. The proposed algorithm, KCDCDynDE, is applicable to static as well as temporal data clustering.

KCDCDynDE adapts CDCDynDE by adding a memory component, replacing the repulsion mechanism and using a convergence measure, the inter-cluster and intra-cluster distance measures and the silhouette validity index.

In order to give algorithms sufficient time to cluster the data, a convergence measure is necessary. A population is considered to have converged if the average distance between the best individual and all other individuals in the population is smaller than a predefined radius r_{conv}. When all the populations have converged, the inter-cluster distance and intra-cluster distance are calculated.

The inter-cluster and intra-cluster distances of the best position are calculated at the start of the algorithm and kept in the algorithm's memory as the previous inter-cluster and previous intra-cluster distances. Once all populations converge, the inter-cluster and intra-cluster distances are calculated again and compared against their previous values. If any of the following conditions are true, a new cluster is added in the form of a new population, where individuals are initialised to random positions within the search space:

- Both the inter-cluster distance and the intra-cluster distance have increased. An increase in both values means that the clusters have become more separated from each other but less compact. This may be caused by a prior merge of two sub-populations, as illustrated in Figures 8.1(a) and 8.1(b), or by the introduction of

a new cluster of data in the environment.

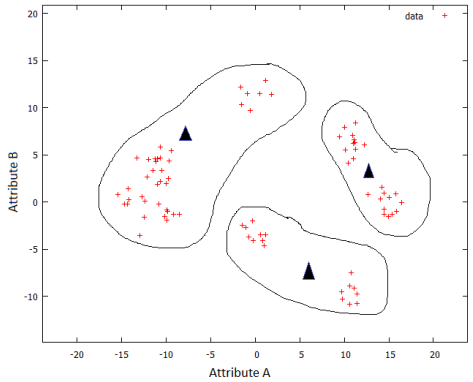
- The intra-cluster distance has decreased and the inter-cluster distance has increased, in other words both values have improved. An improvement in both values means that the clusters are becoming more separate from each other and more compact. This may be caused by a prior addition of a new population, as illustrated in Figures 8.1(c) and 8.1(d), or a change in the environment that caused the dataset to fit the current clustering solution better. Adding a new population may improve the clustering solution further.

- The intra-cluster distance has increased and the inter-cluster distance has decreased, in other words both values have worsened. A degradation in both values means that the clusters are becoming less separated, yet less compact. This may be caused by a prior merge of populations, as illustrated in Figures 8.1(e) and 8.1(f). The addition of a new cluster may encourage the exploration of different areas of the search space in order to improve the clustering solution.

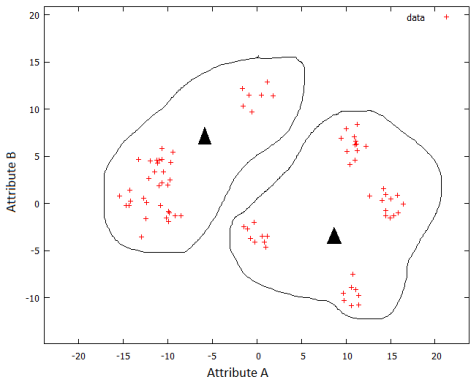
The conditions to add a new cluster have the possibility of being true if a good solution has been found. In order to avoid adding a cluster if a good solution has been found, the silhouette validity index was used as an additional condition, measuring the quality of the current solution. A new cluster may only be added if any of the previous conditions are true and if the silhouette of the solution is below a silhouette threshold. If the silhouette is above a silhouette threshold, no new cluster is added. If a change in the environment occurs, the silhouette of the solution will change and addition of new clusters can continue.

The CDCDynDE is also adapted by replacing repulsion with the merging of populations. If two populations are within r_{excl} from each other, the populations are merged and a randomly selected half of each population is discarded in order to keep the population sizes the same.

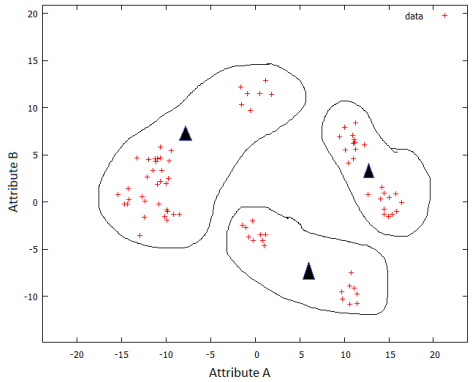
The new population's individuals are re-initialised to random positions in the search space in order to promote exploration. Some clustering algorithms split a cluster when a new cluster is to be added [110][121][128]. This is not viable for the KCDCDynDE as



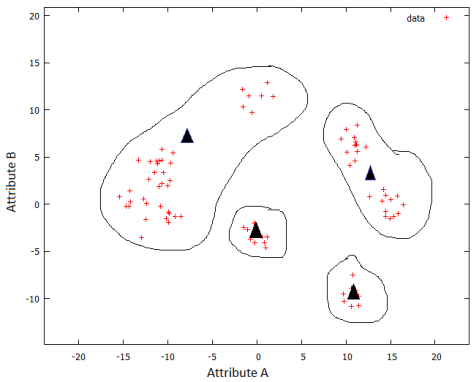
(a) Example A: Previously Iteration



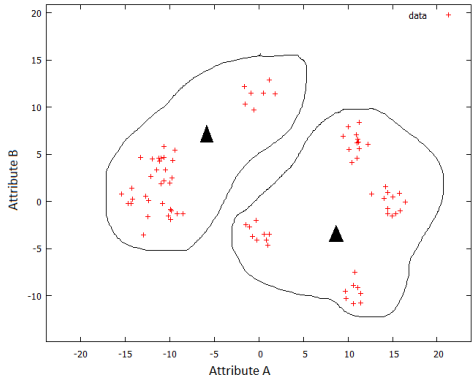
(b) Example A: Current iteration after a merge of two clusters



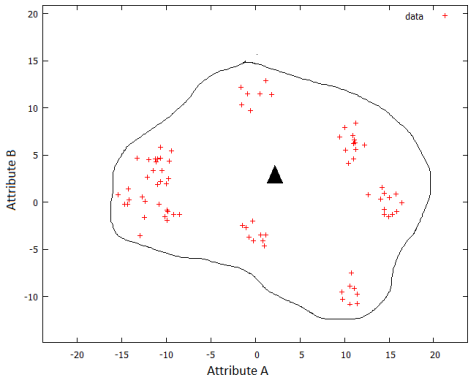
(c) Example B: Previously Iteration



(d) Example B: Current iteration after addition of a cluster



(e) Example C: Previously Iteration



(f) Example C: Current iteration after merge of two clusters into one

Figure 8.1: Examples of Changes That May Require the Addition of a New Cluster

the merging mechanism of the algorithm would remove the new population.

KCDCDynDE adds two new parameters to the DynDE, namely the silhouette threshold and the convergence radius. The silhouette threshold is a problem independent value, chosen as 0.71, based on the conclusions of Kaufman and Rousseeuw [62] summarised in Table 3.1. The influence of the convergence radius on the algorithm has, however, not been investigated and is proposed as future work. Table 8.1 summarises the parameters required by the KCDCDynDE.

Table 8.1: KCDCDynDE Parameters

Parameter	**Origin**	**To be optimised for experiments**
Scale Parameter	DE	yes
Crossover probability	DE	yes
Exclusion Radius	DynDE	yes
% of Brownian individuals	DynDE	yes
Convergence Radius	new	yes
Silhouette threshold	new	no

Algorithm 15 summarises the KCDCDynDE algorithm. The KCDCDynDE algorithm aims to determine K, which is the optimal number of clusters. The algorithm begins by initialising S sub-populations, where $S >= 2$ aiming for $S = K$ in the end.

8.2 Summary

This chapter proposed the KCDCDynDE algorithm, which adapts the CDCDynDE algorithm to determine the number of clusters dynamically.

Algorithm 15 KCDCDynDE

1: Initialise S sub-populations of individuals, where $S >= 2$
2: **while** stopping condition is not reached **do**
3: Generate a context individual
4: Evaluate each sub-population by performing the steps described in lines 4-8 of Algorithm 9 using the context individual
5: Compare the global best individual positions from each sub-population to each other
6: **if** the global best positions of both sub-populations are within r_{excl} of each other **then**
7: Discard half of each population at random
8: Merge the two populations
9: **end if**
10: **if** all populations have converged **then**
11: Calculate the silhouette of the global best position
12: Calculate the new inter-cluster and intra-cluster distances
13: **if** the silhouette < 0.71 AND one of the conditions to add a cluster is true **then**
14: Add a cluster in the form of a new randomly initialised population
15: **end if**
16: Replace the previous inter-cluster and intra-cluster distances with the new values
17: **end if**
18: **for** each sub-population **do**
19: **for** each individual $\mathbf{x}_i$ in the sub-population **do**
20: Update $\mathbf{x}_i$ using lines 10-16 of Algorithm 9
21: **end for**
22: **end for**
23: Make a percentage of the weakest individuals Brownian individuals and update them using equation (6.3)
24: **end while**

Chapter 9

Experimental Set-up and Results

This chapter summarises the results of applying the PSO and DE algorithms described in this thesis to the datasets mentioned in Chapter 4. The results are illustrated using figures and tables, which are then discussed in further detail. There are five sections in this chapter, each dealing with a separate topic. The sections are organised as follows:

- Section 9.1 - **Statistics used for algorithm comparison**: This section summarises the statistical approach used to compare the algorithms to each other.
- Section 9.2 - **Particle Swarm Optimisation Results**: This section summarises the results for the PSO algorithms introduced in Chapter 5.
- Section 9.3 - **Differential Evolution Results**: This section summarises the results for the DE algorithms introduced in Chapter 6.
- Section 9.4 - **Particle Swarm Optimisation Versus Differential Evolution**: This section compares the best performing PSO from Section 9.2 to the best performing DE from Section 9.3.
- Section 9.5 - **K-independent Cooperative Data Clustering Differential Evolution Versus Local Network Neighbourhood Artificial Immune System**: This section summarises the results for the KCDCDynDE algorithm intro-

duced in Chapter 8, and the comparisson of the algorithm against the LNNAIS.

All algorithms were implemented using the Computational Intelligence library (CILib) [22][88], except for the LNNAIS, for which existing results by Graaff and Engelbrecht [46] were used. Experiments were ran on the CiClops [95] cluster available at the University of Pretoria. Lastly, R [98] was used for the statistical analysis of the results.

9.1 Statistics Used for Algorithm Comparison

A wins and losses approach inspired by Helbig and Engelbrecht's [52] technique was used for statistical analysis of the resulting Ray-Turi validity index values. In order for each sample to have a more accurate representation of the Ray-Turi validity index value over time, the Ray-Turi validity indexes were averaged over the number of changes that occurred in the environment before a change in the environment occurs. Each algorithm was compared against each other algorithm for each problem using 30 independent samples. First a Kruskal-Wallis test was used to determine if there is a statistically significant difference between the performance of the two algorithms being compared for a particular problem. If there is a statistically significant difference between the algorithms, a Mann-Whitney U test was performed and the resulting U-values were used to determine the winning and losing algorithm. These tests are performed for every combination of algorithms and all problems. The results reported are only those with statistical significance at a significance level of 0.05 (95% confidence).

9.2 Particle Swarm Optimisation Results

This section reports the results of applying the PSOs described in Chapter 5 to the 225 artificially generated datasets described in Chapter 4. The six PSOs evaluated are the DCPSO, RDCPSO, MDCPSO, CDCPSO, CMDCPSO and elitist cooperative multi-swarm data clustering particle swarm optimisation (eCMDCPSO).

9.2.1 Experimental Set-up

The PSO's populations were initialised within the bounds of the dataset, which are defined per dimension by the lowest and highest value of each attribute in the dataset. Centroids leaving the boundaries of the dataset were re-set to remain on the boundary. The quality of a cluster was determined by each algorithm using the quantization error defined in equation (3.10). The inter-cluster and intra-cluster distances defined in equations (3.1) and (3.3) are reported and the Ray-Turi validity index defined in equation (3.7) is used to compare the algorithms. A population size of 50, 1000 iterations, inertia of 0.729844 [28], and social and cognitive components of 1.496180 [28] were used across all the algorithms, where each experiment was ran for 30 samples. The parameters used for each PSO are summarised in Table 9.1, where the column labelled CMDCPSO summarises the parameters used for both CMDCPSO and the eCMDCPSO.

Table 9.1: Parameters for PSOs

Parameter	**DCPSO**	**RDCPSO**	**MDCPSO**	**CDCPSO**	**CMDCPSO**
Percentage of population re-initialised	N/A	10%	100%	N/A	100%
Exclusion radius	N/A	N/A	1.0	N/A	1.0
Total populations	1	1	8 (K)	8 (K)	9 ($K+1$)

9.2.2 Results and Discussion

The average resulting values for the inter-cluster distance, intra-cluster distance and Ray-Turi validity index are reported in Table 9.2, where the best values are represented in bold. The CDCPSO obtained the highest inter-cluster distance, and therefore the most separate clusters. The DCPSO, on the other hand, resulted in the lowest inter-cluster distance and, therefore, with the smallest distance between clusters.

The MDCPSO resulted in the lowest intra-cluster distance, meaning that the algorithm

Table 9.2: Averages and Standard Deviation for Inter-cluster Distance, Intra-cluster Distance and Ray-Turi Validity for Each PSO Algorithm Over All Environment Types and Dimensions

Algorithm	**Inter-cluster Distance**	**Intra-cluster Distance**	**Ray-Turi Validity**
DCPSO	20.854 ± 2.041	6.634 ± 4.042	1.262 ± 0.369
RDCPSO	21.124 ± 2.383	6.915 ± 4.320	1.254 ± 0.374
CDCPSO	**23.394 ± 3.135**	7.230 ± 4.011	**0.989 ± 0.236**
MDCPSO	20.302 ± 1.734	**5.503 ± 3.543**	1.067 ± 0.115
CMDCPSO	18.901 ± 1.170	9.431 ± 1.977	1.715 ± 0.409
eCMDCPSO	19.399 ± 1.133	9.520 ± 1.946	2.002 ± 0.651

obtained the most compact clusters. The eCMDCPSO, on the other hand obtained the least compact clusters. The rest of the algorithms obtained similar results for the intra-cluster distance which were higher than the eCMDCPSO but lower than the CDCPSO.

Both, the inter-cluster and intra-cluster distances need to be considered in order to determine the quality of a clustering solution. The results discussed so far imply that the CDCPSO results in the most separate but non-compact clusters, while the eCMDCPSO results in the most non-separate but compact clusters. The Ray-Turi validity index is a more accurate representation of the clustering solution as it takes both distance measures into consideration.

The CDCPSO resulted in the lowest Ray-Turi validity index value, while the DCPSO and RDCPSO obtained the highest values. This implies that the DCPSO and RDCPSO performed the temporal data clustering task least effectively, while the CDCPSO performed the task best. The other algorithms, performed the task better than the DCPSO and RDCPSO algorithms, but worse than the CDCPSO.

Figure 9.1 reports the total statistically significant wins and losses obtained by each PSO algorithm, where $\#wins$ is the number of wins, $\#losses$ is the number of losses and $Diff = \#wins - \#losses$. The CDCPSO obtained the most wins and least losses, leading to the largest $Diff$ value. The MDCPSO followed, with the second largest $Diff$

value. The DCPSO and RDCPSO, on the other hand, obtained 33 and 48 more wins than losses respectively. Lastly, the CMDCPSO and eCMDCPSO resulted in negative $Diff$ values and, therefore, significantly more losses than wins.

Attribute	$\#wins$	$\#losses$	$Diff$
DCPSO	458	425	33
RDCPSO	466	418	48
CDCPSO	**996**	**109**	**887**
MDCPSO	814	290	524
CMDCPSO	207	898	-691
eCMDCPSO	152	953	-801

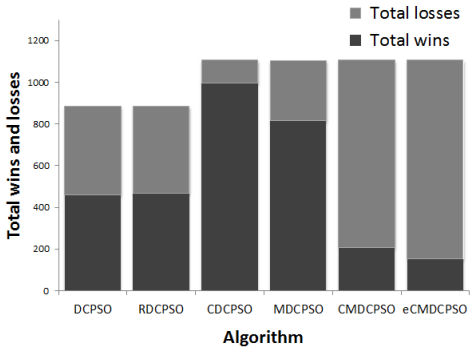


Figure 9.1: Total Statistically Significant Wins and Losses Per PSO Algorithm

Figure 9.2 illustrates the performance of each algorithm on 3, 8 and 15 dimensional problems by depicting the $Diff$ value obtained over the different problems. The results are averaged over all frequencies, severities and migration types. The $Diff$ values were averaged over all severities, frequencies and migration patterns.

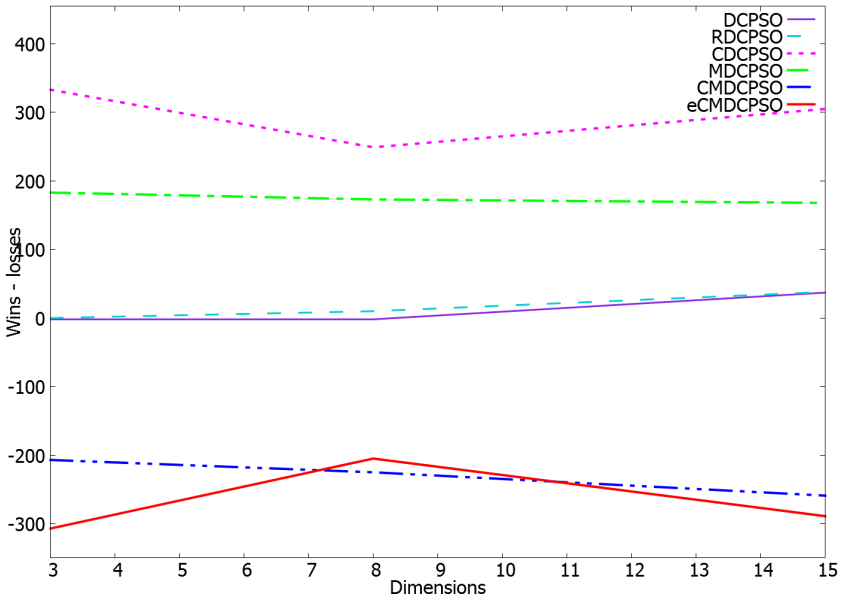


Figure 9.2: $Diff$ Per PSO Algorithm Over All Dimensions

The CDCPSO obtained the largest $Diff$ value over all dimensions, while the eCMDCPSO obtained the lowest $Diff$ value for dimensions 3 and 15. The eCMDCPSO showed improved performance for 8-dimensional problems, decreasing the total wins of the CDCPSO. The CDCPSO and MDCPSO obtained significantly more wins than losses throughout all dimensions, while the other algorithms resulted in either more losses than wins or a small positive $Diff$ value. Lastly, the CMDCPSO demonstrated a decrease in performance as dimensions increased, while the MDCPSO, DCPSO and RDCPSO showed to be insensitive to dimension.

Table 9.3 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each PSO over 3, 8 and 15 dimensional problems. The MDCPSO obtained the lowest average intra-cluster distance over all dimensions. CDCPSO, however, obtained not only the highest average inter-cluster distance, but also the lowest average Ray-Turi validity index over all dimensions, which shows superior performance across all dimensions.

Table 9.3: Average PSO Results by Dimension

D	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
3	Inter-cluster distance	20.752 ± (0.217)	20.743 ± (0.183)	**21.694 ± (0.607)**	20.038 ± (0.929)	18.495 ± (0.621)	19.335 ± (0.625)
	Intra-cluster distance	3.623 ± (1.358)	3.664 ± (1.406)	3.815 ± (1.753)	**3.063 ± (0.928)**	7.692 ± (1.296)	7.811 ± (0.645)
	Ray-Turi validity index	1.411 ± (0.437)	1.433 ± (0.458)	**0.920 ± (0.327)**	1.090 ± (0.145)	1.895 ± (0.307)	2.410 ± (0.447)
	$Diff$	-2	0	**333**	183	-207	-307
8	Inter-cluster distance	19.805 ± (2.193)	20.145 ± (2.535)	**22.236 ± (2.645)**	19.521 ± (1.835)	19.020 ± (1.178)	19.418 ± (1.282)
	Intra-cluster distance	7.194 ± (3.923)	7.553 ± (4.204)	7.450 ± (3.523)	**5.903 ± (3.520)**	10.261 ± (1.622)	10.393 ± (1.795)

Table 9.3: Average PSO Results by Dimension

D	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
	Ray-Turi validity index	1.299 ± (0.374)	1.275 ± (0.344)	**1.097 ± (0.189)**	1.112 ± (0.092)	1.631 ± (0.417)	1.742 ± (0.581)
	$Diff$	-2	10	**249**	173	-225	-205
15	Inter-cluster distance	22.007 ± (2.268)	22.485 ± (2.748)	**26.252 ± (3.098)**	21.345 ± (1.728)	19.190 ± (1.434)	19.443 ± (1.340)
	Intra-cluster distance	9.084 ± (4.022)	9.529 ± (4.276)	10.424 ± (3.265)	**7.544 ± (3.735)**	10.341 ± (1.678)	10.356 ± (1.813)
	Ray-Turi validity index	1.075 ± (0.132)	1.053 ± (0.133)	**0.951 ± (0.080)**	0.997 ± (0.050)	1.619 ± (0.428)	1.855 ± (0.688)
	$Diff$	37	38	**305**	168	-259	-289

Figure 9.3 illustrates the $Diff$ value of each algorithm on problems of change frequency 1, 2, 3, 4 and 5. The results were averaged over all severities, dimensions and migration types. The CDCPSO, once again, resulted in the largest $Diff$ value over all frequencies, followed by the MDCPSO. Both algorithms were shown to be insensitive to change frequency, while the performance of the DCPSO and RDCPSO declined as changes became more frequent (in other words, when the frequency value lowers).

Table 9.4 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each PSO over problems of frequencies 1 to 5. The observations across all frequencies were the same as across all dimensions.

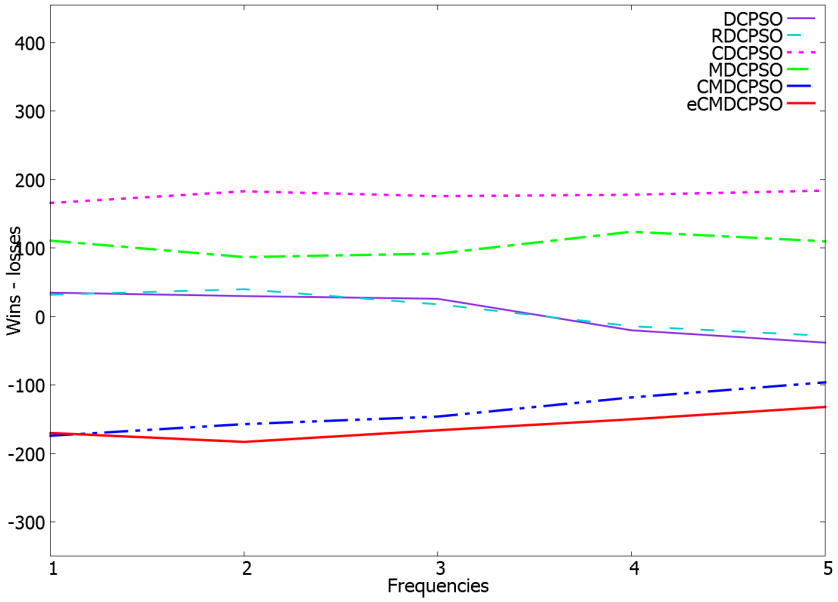


Figure 9.3: $Diff$ Per PSO Algorithm Over All Frequencies

Table 9.4: Average PSO Results by Frequency

F	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
1	Inter-cluster distance	21.683 ± (2.820)	21.881 ± (3.228)	**24.761 ± (3.730)**	20.791 ± (2.417)	19.104 ± (1.230)	19.551 ± (1.135)
	Intra-cluster distance	7.888 ± (4.853)	8.011 ± (4.951)	8.337 ± (4.217)	**6.529 ± (4.418)**	9.304 ± (2.069)	9.679 ± (1.879)
	Ray-Turi validity index	1.158 ± (0.217)	1.144 ± (0.254)	**0.991 ± (0.190)**	1.054 ± (0.111)	1.720 ± (0.421)	1.841 ± (0.531)
	$Diff$	35	32	**166**	111	-174	-170
2	Inter-cluster distance	21.144 ± (2.152)	21.470 ± (2.522)	**23.669 ± (3.358)**	20.513 ± (1.907)	18.975 ± (1.201)	19.488 ± (1.103)
	Intra-cluster distance	6.964 ± (4.255)	7.379 ± (4.553)	7.296 ± (4.042)	**5.968 ± (3.923)**	9.341 ± (2.030)	9.566 ± (1.908)

Table 9.4: Average PSO Results by Frequency

F	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
	Ray-Turi validity index	1.209 ± (0.354)	1.216 ± (0.355)	**0.957 ± (0.206)**	1.043 ± (0.075)	1.714 ± (0.394)	1.989 ± (0.639)
	Diff	30	40	**183**	87	-157	-183
3	Inter-cluster distance	20.850 ± (1.694)	21.032 ± (1.934)	**23.246 ± (2.871)**	20.233 ± (1.527)	18.917 ± (1.194)	19.445 ± (1.196)
	Intra-cluster distance	6.440 ± (3.840)	6.758 ± (4.025)	7.116 ± (3.881)	**5.683 ± (3.433)**	9.596 ± (1.964)	9.441 ± (1.925)
	Ray-Turi validity index	1.215 ± (0.344)	1.225 ± (0.341)	**0.972 ± (0.216)**	1.069 ± (0.113)	1.719 ± (0.436)	2.034 ± (0.705)
	Diff	26	18	**176**	92	-146	-166
4	Inter-cluster distance	20.518 ± (1.572)	20.835 ± (2.072)	**22.792 ± (2.629)**	20.123 ± (1.304)	18.821 ± (1.147)	19.392 ± (1.104)
	Intra-cluster distance	6.173 ± (3.654)	6.497 ± (4.070)	6.989 ± (3.989)	**5.317 ± (3.046)**	9.468 ± (1.869)	9.480 ± (1.989)
	Ray-Turi validity index	1.290 ± (0.319)	1.288 ± (0.360)	**0.996 ± (0.242)**	1.081 ± (0.145)	1.723 ± (0.420)	2.060 ± (0.672)
	Diff	-20	-14	**178**	124	-118	-150
5	Inter-cluster distance	20.078 ± (1.151)	20.402 ± (1.466)	**22.501 ± (2.339)**	19.848 ± (0.967)	18.689 ± (1.013)	19.118 ± (1.061)
	Intra-cluster distance	5.704 ± (2.962)	5.933 ± (3.511)	6.410 ± (3.609)	**4.020 ± (1.726)**	9.448 ± (1.910)	9.434 ± (1.992)
	Ray-Turi validity index	1.436 ± (0.488)	1.395 ± (0.475)	**1.030 ± (0.302)**	1.085 ± (0.113)	1.697 ± (0.366)	2.088 ± (0.658)

Table 9.4: Average PSO Results by Frequency

F	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
	$Diff$	-38	-28	**184**	110	-96	-132

Figure 9.4 illustrates the performance of each algorithm on problems of severity 1, 2, 3, 4 and 5. The results were averaged over all frequencies, dimensions and migration types. The observations across the various severities are the same as over various frequencies. The CDCPSO was shown to be insensitive to the severity of change, while the MDCPSO shows a slight improvement in performance as severities increase.

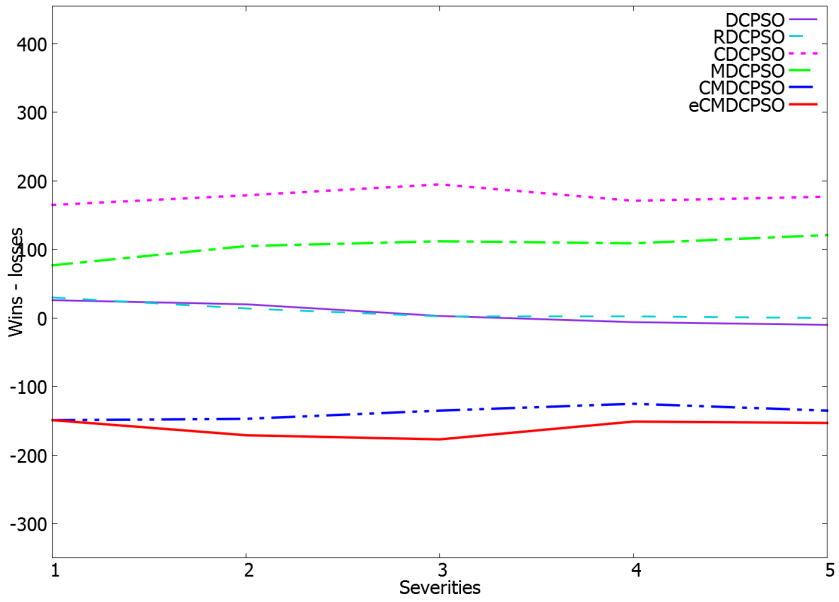


Figure 9.4: $Diff$ Per PSO Algorithm Over All Severities

Table 9.5 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each PSO over problems of severities 1 to 5. The same observations are seen across all severities as are seen across all frequencies.

Table 9.5: Average PSO Results by Severity

S	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
1	Inter-cluster distance	20.478 ± (1.650)	20.572 ± (1.620)	**22.599 ± (3.101)**	20.143 ± (1.348)	18.652 ± (1.218)	19.349 ± (1.148)
	Intra-cluster distance	5.785 ± (3.264)	5.853 ± (3.475)	6.453 ± (3.926)	**4.762 ± (2.793)**	9.279 ± (1.998)	9.401 ± (1.972)
	Ray-Turi validity index	1.217 ± (0.240)	1.184 ± (0.257)	**0.996 ± (0.174)**	1.084 ± (0.095)	1.766 ± (0.430)	1.945 ± (0.609)
	$Diff$	26	30	**165**	77	-149	-149
2	Inter-cluster distance	20.914 ± (2.108)	21.218 ± (2.660)	**23.347 ± (3.234)**	20.250 ± (1.674)	18.846 ± (1.068)	19.351 ± (1.080)
	Intra-cluster distance	6.562 ± (4.265)	6.933 ± (4.654)	7.204 ± (4.166)	**5.197 ± (3.382)**	9.258 ± (2.011)	9.423 ± (1.894)
	Ray-Turi validity index	1.240 ± (0.383)	1.271 ± (0.365)	**0.991 ± (0.234)**	1.073 ± (0.110)	1.716 ± (0.406)	2.044 ± (0.705)
	$Diff$	20	14	**179**	105	-147	-171
3	Inter-cluster distance	20.833 ± (2.143)	21.123 ± (2.475)	**23.507 ± (3.065)**	20.519 ± (1.929)	18.958 ± (1.200)	19.193 ± (1.131)
	Intra-cluster distance	6.682 ± (4.016)	6.875 ± (4.156)	7.141 ± (3.830)	**5.797 ± (4.021)**	9.209 ± (2.013)	9.582 ± (1.904)
	Ray-Turi validity index	1.279 ± (0.402)	1.261 ± (0.401)	**0.937 ± (0.217)**	1.052 ± (0.101)	1.726 ± (0.393)	2.003 ± (0.603)
	$Diff$	3	2	**195**	112	-135	-177
4	Inter-cluster distance	20.962 ± (2.039)	21.267 ± (2.309)	**23.597 ± (2.871)**	20.232 ± (1.708)	19.028 ± (1.142)	19.477 ± (1.165)

Table 9.5: Average PSO Results by Severity

S	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
	Intra-cluster distance	6.959 ± (4.127)	7.396 ± (4.372)	7.585 ± (4.016)	**5.824 ± (3.564)**	9.609 ± (1.786)	9.644 ± (1.962)
	Ray-Turi validity index	1.280 ± (0.410)	1.289 ± (0.435)	**1.018 ± (0.288)**	1.066 ± (0.126)	1.680 ± (0.386)	2.063 ± (0.697)
	$Diff$	-6	2	**171**	109	-125	-151
5	Inter-cluster distance	21.087 ± (2.147)	21.440 ± (2.585)	**23.920 ± (3.203)**	20.364 ± (1.905)	19.023 ± (1.163)	19.624 ± (1.081)
	Intra-cluster distance	7.180 ± (4.263)	7.520 ± (4.589)	7.765 ± (3.938)	**5.937 ± (3.654)**	9.804 ± (1.977)	9.551 ± (1.962)
	Ray-Turi validity index	1.292 ± (0.375)	1.263 ± (0.376)	**1.005 ± (0.244)**	1.059 ± (0.134)	1.686 ± (0.418)	1.956 ± (0.618)
	$Diff$	-10	0	**177**	121	-135	-153

Figure 9.5 illustrates the total wins and losses for each algorithm per migration type, where the three migration types are pattern migration, centroid migration and cluster migration. The results are averaged over all frequencies, severities and dimensions. The CDCPSO obtained the most wins and least losses on pattern migration and cluster migration problems, while the CMDCPSO obtained very few wins. The MDCPSO, on the other hand, obtained more wins than losses for all three migration types. The MDCPSO obtained the best results for centroid migration problems, but was outperformed by the CMDCPSO for pattern and cluster migration problems. Lastly, the RDCPSO and DCPSO obtained more wins than the eCMDCPSO and CMDCPSO for pattern and cluster migration problems, but significantly dropped performance for centroid migration problems.

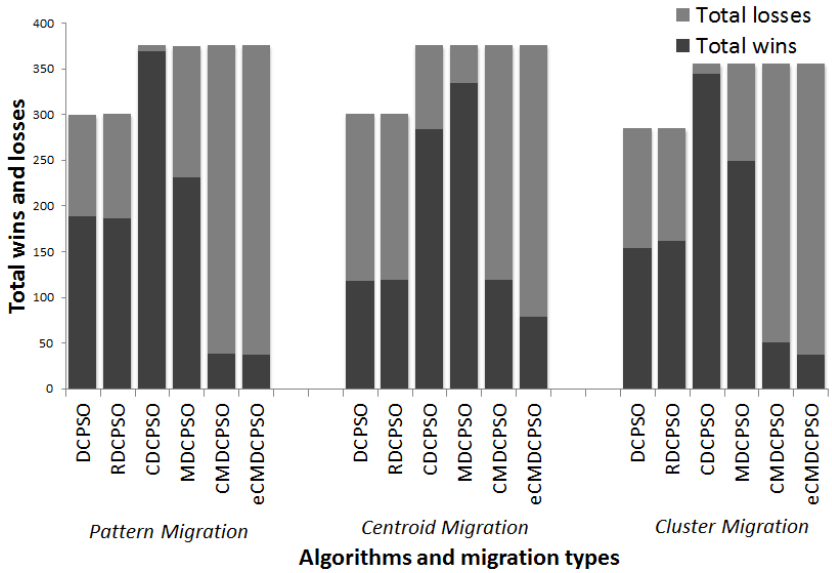


Figure 9.5: Wins and Losses Per PSO Algorithm Over All Migration Types

Table 9.6 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each PSO over pattern migration, cluster migration and centroid migration problems. The CDCPSO and the MDCPSO obtained similar results over the migration types as over the various frequencies, severities and dimensions with regards to the inter-cluster, intra-cluster and Ray-Turi values. The one difference is that the MDCPSO's Ray-Turi validity index outperformed the CDCPSO's validity index for cluster migration problems.

Table 9.6: Average PSO Results by Migration Type

M	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
Pattern	Inter-cluster distance	20.004 ± (0.999)	20.036 ± (1.016)	**21.778 ± (1.587)**	19.982 ± (1.034)	18.431 ± (0.575)	18.884 ± (0.648)
	Intra-cluster distance	4.371 ± (1.548)	4.603 ± (1.734)	5.037 ± (2.307)	**3.770 ± (0.941)**	8.354 ± (1.547)	8.703 ± (1.466)
	Ray-Turi validity index	1.094 ± (0.113)	1.108 ± (0.110)	**0.895 ± (0.157)**	1.067 ± (0.096)	1.795 ± (0.341)	2.030 ± (0.518)

Table 9.6: Average PSO Results by Migration Type

M	Measure	DCPSO	RDCPSO	CDCPSO	MDCPSO	CMDCPSO	eCMDCPSO
	Diff	77	72	**363**	88	-299	-301
Centroid	Inter-cluster distance	20.077 ± (1.073)	20.217 ± (1.119)	**22.720 ± (2.155)**	19.666 ± (0.987)	19.100 ± (1.169)	19.602 ± (1.099)
	Intra-cluster distance	4.921 ± (2.004)	5.095 ± (2.249)	5.732 ± (2.565)	**4.301 ± (1.888)**	9.336 ± (2.109)	9.406 ± (2.111)
	Ray-Turi validity index	1.138 ± (0.135)	1.116 ± (0.141)	**0.876 ± (0.149)**	1.037 ± (0.094)	1.806 ± (0.431)	2.121 ± (0.725)
	Diff	-66	-62	191	**293**	-137	-219
Cluster	Inter-cluster distance	22.483 ± (2.512)	23.119 ± (2.948)	**25.684 ± (3.727)**	21.256 ± (2.350)	19.173 ± (1.435)	19.710 ± (1.344)
	Intra-cluster distance	10.609 ± (4.304)	11.048 ± (4.683)	10.920 ± (3.935)	**8.440 ± (4.468)**	10.604 ± (1.514)	10.453 ± (1.776)
	Ray-Turi validity index	1.553 ± (0.498)	1.537 ± (0.515)	1.197 ± (0.236)	**1.095 ± (0.140)**	1.543 ± (0.392)	1.856 ± (0.660)
	Diff	22	38	**333**	143	-255	-281

In summary, the Ray-Turi validity index showed that on average, over all problems, the CDCPSO algorithm performed the clustering task best by obtaining the most wins and least losses over the various environmental changes. Both CMDCPSO and eCMDCPSO, on the other hand, showed the worst performance across all the environmental changes. It is hypothesised that the reason for the failure of the CMDCPSO and eCMDCPSO to perform the clustering task may have been the anti-convergence component. The anti-convergence required the algorithms to have an extra population exploring the environment, which replaces the population that was re-initialised when all populations

converge. This extra population was added to remedy the possibility of a final solution containing a recently re-initialised centroid. However, as this population explored the search space regardless of the context, the centroid it contributes to the context particle, once its status changes to non-explorer, may be obsolete or may already exist in the context particle.

9.3 Differential Evolution Results

This section reports the results of applying the DEs described in Chapter 6 to the same 225 artificially generated datasets clustered by the PSOs. The four DEs evaluated are DCDE, RDCDE, DCDynDE and CDCDynDE.

9.3.1 Experimental Set-up

The populations were initialised within the bounds of the dataset and were kept within the boundaries of the dataset in the same way as for the PSOs. The same measures used to analyse the PSOs were used to analyse the DEs, along with the same population size, iterations and total samples. A scaling factor and crossover probability of 0.5 was used for all the DE algorithms. The parameters used for each DE are summarised in Table 9.7.

Table 9.7: Parameters for DEs

Parameter	**DCDE**	**RDCDE**	**DCDynDE**	**CDCDynDE**
Percentage of population re-initialised	N/A	10%	100%	100%
Percentage of Brownian individuals	N/A	N/A	10%	10%
Exclusion radius	N/A	N/A	5.0	5.0
Total populations	1	1	8 (K)	8 (K)

9.3.2 Results and Discussion

Table 9.8 reports average values for the inter-cluster distance, intra-cluster distance and Ray-Turi validity index of the various DEs. The DCDynDE resulted in the lowest inter-cluster distance, while the CDCDynDE obtained the highest value for this measure. The standard deviation of the CDCDynDE inter-cluster distance is a large value, unlike that of the DCDynDE, which implies that the inter-cluster distances varied significantly for the datasets clustered.

Table 9.8: Averages and Standard Deviation for Inter-cluster Distance, Intra-cluster Distance and Ray-Turi Validity for Each DE Algorithm

Algorithm	**Inter-cluster Distance**	**Intra-cluster Distance**	**Ray-Turi Validity**
DCDE	22.983 ± 3.266	10.679 ± 3.981	1.141 ± 0.178
RDCDE	22.963 ± 3.257	10.682 ± 3.979	1.154 ± 0.198
DCDynDE	19.830 ± 3.346	**3.877 ± 2.995**	0.979 ± 0.207
CDCDynDE	**30.072 ± 11.323**	10.39107 ± 8.908	**0.677 ± 0.276**

The DCDynDE obtained the lowest intra-cluster distance. The CDCDynDE, on the other hand, resulted in similar intra-cluster distance to the DCDE and RDCDE, where the RDCDE obtained the highest value.

The results discussed so far imply that the CDCDynDE results in separate but non-compact clusters, while the DCDynDE results in non-separate but compact clusters. The CDCDynDE resulted in the lowest Ray-Turi validity index value, while the DCDE and RDCDE obtained the highest value. This implies that the DCDE and RDCDE performed the temporal data clustering task least effectively, while the CDCDynDE, performed the task best. Lastly, the DCDynDE performed the task better than the DCDE and RDCDE algorithms, but worse than the CDCDynDE.

Figure 9.6 reports the total significant wins and losses obtained by each algorithm.

Attribute	$\#wins$	$\#losses$	$Diff$
DCDE	203	466	-263
RDCDE	204	466	-262
DCDynDE	300	370	-70
CDCDynDE	**632**	**37**	**595**

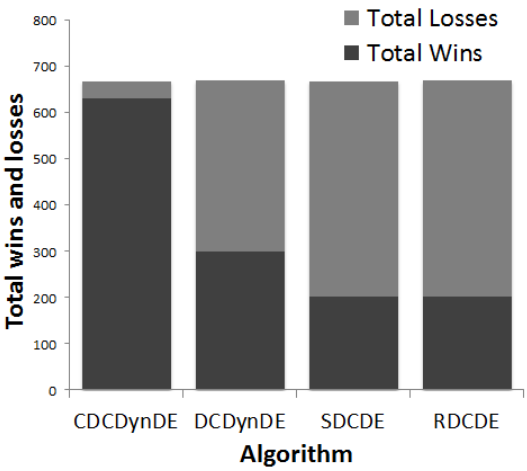


Figure 9.6: Total Statistically Significant Wins and Losses Per DE Algorithm

The CDCDynDE obtained the only positive $Diff$ value along with the most wins and least losses when compared to the other algorithms. The DCDE and RDCDE, on the other hand, obtained significantly more losses than wins leaving the algorithms with very low $Diff$ values. Lastly, the DCDynDE resulted in 70 more losses than wins and therefore in a negative $Diff$ value as well.

Figure 9.7 illustrates the $Diff$ value of each algorithm on 3, 8 and 15 dimensional problems. For three dimensional problems both DCDynDE and CDCDynDE obtained significantly more wins than losses, while DCDE and RDCDE obtained a negative $Diff$ value. The CDCDynDE showed to be insensitive to increasing dimensions, while the DCDynDE's wins are significantly lower as dimensions increase. The DCDE and RDCDE's perform better than the DCDynDE as dimensions increase, but remain with more losses than wins for all dimensions.

Table 9.9 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each DE over the various dimensions. The average inter-cluster, intra-cluster and Ray-Turi values in each dimension match the results of the averages discussed in Table 9.8.

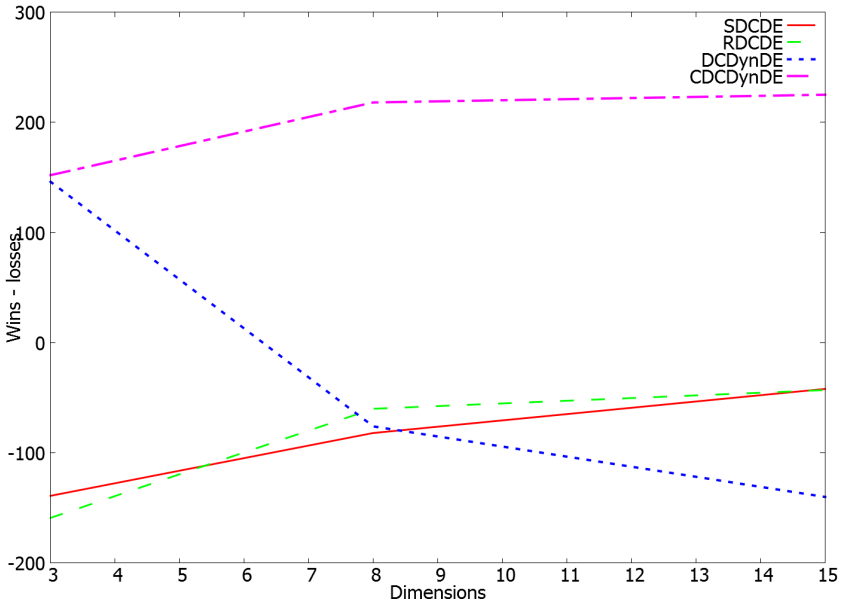


Figure 9.7: $Diff$ Per DE Algorithm Over All Dimensions

Table 9.9: Average DE Results by Dimension

D	Measure	DCDE	RDCDE	DCDynDE	CDCDynDE
3	Inter-cluster distance	20.894 ±(0.622)	20.870 ±(0.660)	19.640 ±(2.049)	**24.887 ±(3.702)**
	Intra-cluster distance	6.768 ±(1.108)	6.802 ±(1.196)	**1.934 ±(0.807)**	5.569 ±(3.824)
	Ray-Turi validity index	1.397 ±(0.189)	1.434 ±(0.231)	0.757 ±(0.262)	**0.657 ±(0.414)**
	$Diff$	-159	-139	146	**152**
8	Inter-cluster distance	23.404 ±(3.762)	23.429 ±(3.788)	19.155 ±(3.059)	**29.502 ±(9.298)**
	Intra-cluster distance	12.256 ±(3.282)	12.276 ±(3.283)	**4.099 ±(2.704)**	10.335 ±(7.276)
	Ray-Turi validity index	1.097 ±(0.048)	1.093 ±(0.052)	1.095 ±(0.096)	**0.654 ±(0.210)**

Table 9.9: Average DE Results by Dimension

D	Measure	DCDE	RDCDE	DCDynDE	CDCDynDE
	$Diff$	-60	-82	-76	**218**
15	Inter-cluster distance	25.796 ±(3.791)	25.745 ±(3.777)	20.593 ±(3.874)	**36.216 ±(14.800)**
	Intra-cluster distance	14.526 ±(3.191)	14.498 ±(3.174)	**5.491 ±(3.241)**	15.540 ±(10.960)
	Ray-Turi validity index	1.011 ±(0.035)	1.013 ±(0.029)	1.067 ±(0.090)	**0.733 ±(0.084)**
	$Diff$	-43	-42	-140	**225**

Figure 9.8 illustrates the performance of each algorithm on problems of frequencies 1, 2, 3, 4 and 5.

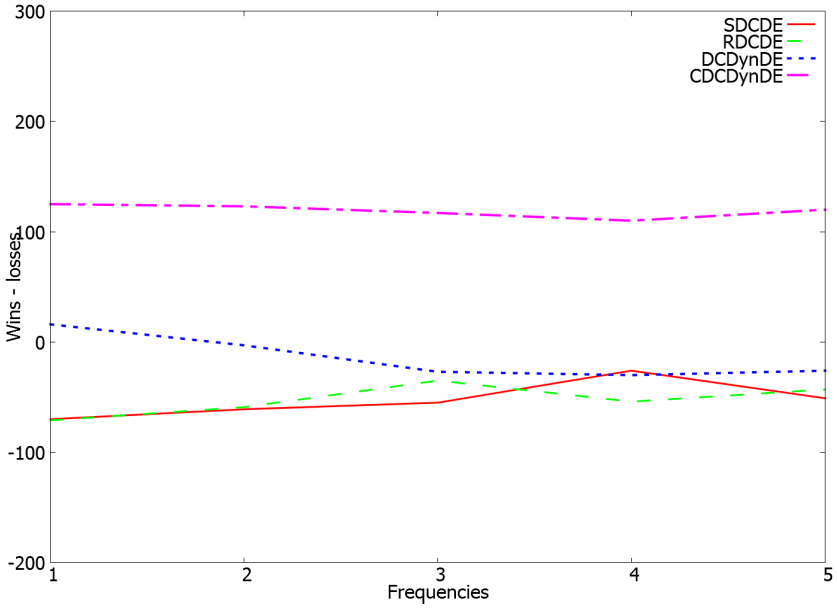


Figure 9.8: $Diff$ Per DE Algorithm Over All Frequencies

The CDCDynDE maintained a large positive $Diff$ value through all five frequencies, while the DCDynDE performed better in environments with more frequent changes (i.e. a frequency of 1) than in environments with less changes. The DCDE and RDCDE handled environments with less changes (frequencies 3 to 5) better than those changing more frequently (frequencies 1 to 2).

Table 9.10 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each DE over problems of frequencies 1 to 5. The observed trends per frequency are the same as the observed trends per dimension.

Table 9.10: Average DE Results by Frequency

F	Measure	DCDE	RDCDE	DCDynDE	CDCDynDE
1	Inter-cluster distance	24.493 ± (4.435)	24.425 ± (4.532)	19.672 ± (3.600)	**31.057 ± (10.695)**
	Intra-cluster distance	12.229 ± (4.733)	12.282 ± (4.748)	**4.426 ± (3.368)**	11.414 ± (8.505)
	Ray-Turi validity index	1.208 ± (0.227)	1.210 ± (0.239)	1.000 ± (0.178)	**0.743 ± (0.233)**
	$Diff$	-71	-70	16	**125**
2	Inter-cluster distance	23.786 ± (4.042)	23.845 ± (4.112)	20.439 ± (4.964)	**31.422 ± (12.065)**
	Intra-cluster distance	11.619 ± (4.558)	11.631 ± (4.539)	**4.360 ± (4.286)**	11.480 ± (9.421)
	Ray-Turi validity index	1.171 ± (0.209)	1.198 ± (0.238)	0.971 ± (0.187)	**0.708 ± (0.277)**
	$Diff$	-59	-61	-3	**123**
3	Inter-cluster distance	23.430 ± (3.853)	23.376 ± (3.705)	19.530 ± (2.788)	**31.125 ± (11.171)**

Table 9.10: Average DE Results by Frequency

F	Measure	DCDE	RDCDE	DCDynDE	CDCDynDE
	Intra-cluster distance	11.227 ± (4.237)	11.213 ± (4.199)	**3.695 ± (2.577)**	11.152 ± (8.847)
	Ray-Turi validity index	1.170 ± (0.205)	1.168 ± (0.226)	0.966 ± (0.254)	**0.691 ± (0.272)**
	Diff	-35	-55	-27	**117**
4	Inter-cluster distance	23.002 ± (3.156)	22.976 ± (3.166)	19.523 ± (1.500)	**29.107 ± (10.926)**
	Intra-cluster distance	10.937 ± (3.913)	10.926 ± (3.880)	**3.523 ± (1.606)**	9.696 ± (8.712)
	Ray-Turi validity index	1.164 ± (0.197)	1.194 ± (0.253)	0.982 ± (0.250)	**0.652 ± (0.289)**
	Diff	-54	-26	-30	**110**
5	Inter-cluster distance	22.111 ± (2.062)	22.119 ± (1.995)	19.816 ± (1.092)	**28.297 ± (11.321)**
	Intra-cluster distance	9.905 ± (3.253)	9.908 ± (3.238)	**3.202 ± (1.192)**	8.664 ± (8.616)
	Ray-Turi validity index	1.127 ± (0.152)	1.129 ± (0.169)	0.947 ± (0.256)	**0.614 ± (0.280)**
	Diff	-43	-51	-26	**120**

Figure 9.9 illustrates the performance of each algorithm on problems of severity 1 to 5. The CDCDynDE showed to not be affected by the severity of change, while the other three algorithms were sensitive to the severity.

Table 9.11 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi

validity index for each DE over problems of severities 1 to 5. The trends observed over all severities of change are the same as the trends observed across all frequencies of change.

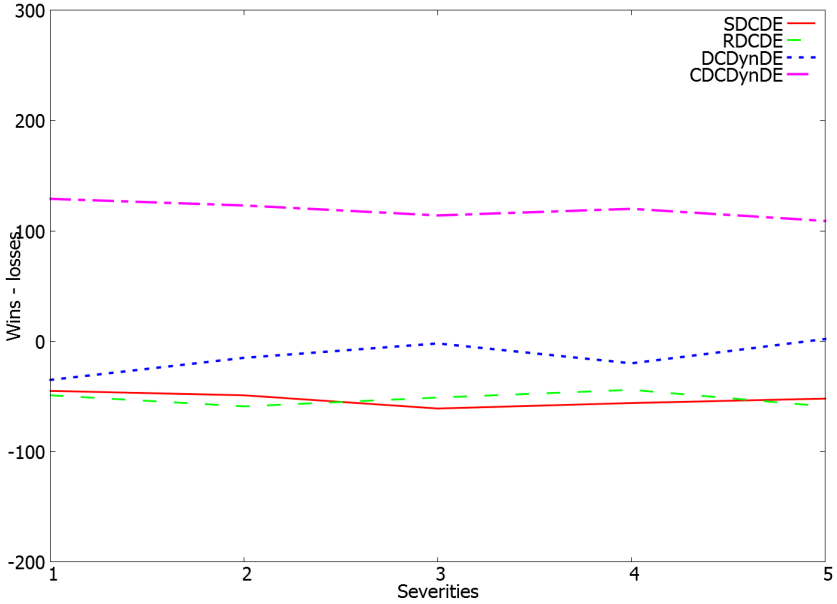


Figure 9.9: $Diff$ per DE algorithm over all severities

Table 9.11: Average DE Results by Severity

S	Measure	DCDE	RDCDE	DCDynDE	CDCDynDE
1	Inter-cluster distance	22.550 ± (3.117)	22.532 ± (3.103)	20.315 ± (3.152)	**23.349 ± (4.828)**
	Intra-cluster distance	10.193 ± (3.901)	10.214 ± (3.923)	**3.753 ± (2.913)**	4.919 ± (4.181)
	Ray-Turi validity index	1.109 ± (0.136)	1.127 ± (0.155)	0.919 ± (0.251)	**0.500 ± (0.245)**
	$Diff$	-49	-45	-35	**129**
2	Inter-cluster distance	23.248 ± (3.876)	23.241 ± (3.913)	19.540 ± (1.994)	**26.837 ± (7.769)**

Table 9.11: Average DE Results by Severity

S	Measure	DCDE	RDCDE	DCDynDE	CDCDynDE
	Intra-cluster distance	10.984 ± (4.419)	10.969 ± (4.411)	**3.688 ± (2.120)**	8.013 ± (6.356)
	Ray-Turi validity index	1.148 ± (0.170)	1.170 ± (0.218)	0.973 ± (0.230)	**0.650 ± (0.239)**
	Diff	-59	-49	-15	**123**
3	Inter-cluster distance	23.540 ± (3.873)	23.527 ± (3.830)	20.202 ± (4.569)	**32.448 ± (12.204)**
	Intra-cluster distance	11.333 ± (4.391)	11.355 ± (4.350)	**4.297 ± (3.931)**	12.325 ± (9.531)
	Ray-Turi validity index	1.185 ± (0.218)	1.185 ± (0.232)	0.992 ± (0.258)	**0.738 ± (0.264)**
	Diff	-51	-61	-2	**114**
4	Inter-cluster distance	23.608 ± (3.513)	23.604 ± (3.568)	19.154 ± (1.542)	**33.833 ± (12.311)**
	Intra-cluster distance	11.554 ± (4.017)	11.572 ± (4.015)	**3.595 ± (1.725)**	13.303 ± (9.546)
	Ray-Turi validity index	1.204 ± (0.235)	1.198 ± (0.238)	0.989 ± (0.198)	**0.754 ± (0.267)**
	Diff	-44	-56	-20	**120**
5	Inter-cluster distance	23.876 ± (3.886)	23.836 ± (3.858)	19.769 ± (3.362)	**34.542 ± (12.675)**
	Intra-cluster distance	11.852 ± (4.267)	11.849 ± (4.233)	**3.872 ± (3.128)**	13.846 ± (9.807)

Table 9.11: Average DE Results by Severity

S	Measure	DCDE	RDCDE	DCDynDE	CDCDynDE
	Ray-Turi validity index	1.196 ± (0.214)	1.219 ± (0.273)	0.992 ± (0.188)	**0.764 ± (0.265)**
	Diff	-59	-52	2	**109**

Figure 9.10 illustrates the total wins and losses for each algorithm per migration type. The CDCDynDE obtained significantly more wins than losses for all migration types. The DCDynDE, on the other hand performed better than the DCDE and RDCDE for centroid and cluster migration problems, but not for pattern migration. Lastly, the DCDE performed better than the RDCDE on pattern migration problems, but worse on cluster and centroid migration problems.

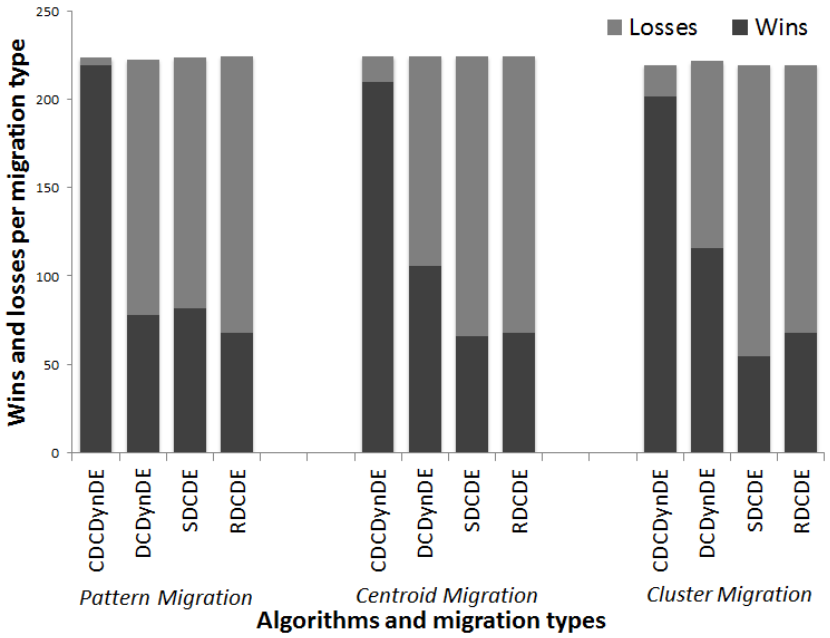


Figure 9.10: *Diff* Per DE Algorithm Over All Migration Types

Table 9.12 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each DE the various migration types. The observed trends are once again

the same across all migration types as across all frequencies, severities and dimensions.

Table 9.12: Average DE Results By Migration Type

M	Measure	DCDE	RDCDE	DCDynDE	CDCDynDE
Pattern	Inter-cluster distance	21.594 ±(1.066)	21.617 ±(1.070)	20.175 ±(1.090)	**22.237 ±(1.949)**
	Intra-cluster distance	9.223 ±(2.559)	9.216 ±(2.563)	**3.243 ±(1.132)**	3.840 ±(2.182)
	Ray-Turi validity index	1.099 ±(0.124)	1.094 ±(0.131)	0.899 ±(0.293)	**0.454 ±(0.221)**
	Diff	-89	-60	-67	**216**
Centroid	Inter-cluster distance	22.072 ±(1.559)	22.075 ±(1.508)	19.337 ±(1.188)	**32.345 ±(11.825)**
	Intra-cluster distance	10.277 ±(2.744)	10.293 ±(2.715)	**3.098 ±(1.164)**	12.410 ±(9.019)
	Ray-Turi validity index	1.191 ±(0.218)	1.201 ±(0.237)	0.975 ±(0.159)	**0.769 ±(0.223)**
	Diff	-89	-93	-13	**195**
Cluster	Inter-cluster distance	26.428 ±(4.814)	26.353 ±(4.886)	19.876 ±(5.165)	**36.023 ±(11.771)**
	Intra-cluster distance	14.050 ±(5.200)	14.067 ±(5.171)	**5.182 ±(4.422)**	15.194 ±(9.017)
	Ray-Turi validity index	1.214 ±(0.226)	1.243 ±(0.268)	1.045 ±(0.186)	**0.821 ±(0.220)**
	Diff	-84	-110	10	**184**

To summarise, the Ray-Turi validity index showed that, on average, the CDCDynDE

algorithm performed the clustering task best. The algorithm obtained the best results regardless of migration types and changes in dimension, frequency or severity. The DCDynDE was highly affected by an increase in dimension, and the DCDE and RDCDE had the worst performance.

9.4 Particle Swarm Optimisation Versus Differential Evolution

This section compares the results of the best PSO selected from Section 9.2 and the best DE selected from Section 9.3. The algorithms compared are the CDCPSO and CDCDynDE.

9.4.1 Experimental Set-up

An adapted F-race algorithm [7] described by Nepomuceno and Engelbrecht [84] was used to optimise the parameters of the CDCPSO and CDCDynDE algorithms, but a single F-race iteration was performed. Parameters were optimised for three problems of each migration type. The problems were selected according to their difficulty, where a problem with a lower dimension, a lower severity and a larger frequency value is considered an easier problem than a problem with a higher dimension, a higher severity and a lower frequency value. The problems over which parameters were optimised are summarised in Table 9.13. Table 9.14 summarises the F-race parameters.

Table 9.15 summarises the parameters used for the CDCPSO and Table 9.16 summarises the parameters used for the CDCDynDE.F-race discarded no parameters when applied to the CDCPSO, implying that the algorithm's performance is not affected by parameter changes. For this reason, the parameters used for the CDCPSO remained the same as in Section 9.2.

F-race discarded 29 parameters when applied to the CDCDynDE. The parameters used

Table 9.13: Problems Used to Tune the Parameters of the CDCPSO and CDCDynDE, where d is the Dimension, f is the Frequency and s is the Severity

Difficulty level	**Pattern Migration**			**Centroid Migration**			**Cluster Migration**		
	d	**f**	**s**	**d**	**f**	**s**	**d**	**f**	**s**
Low	3	5	1	3	5	1	3	5	1
Medium	8	3	3	8	3	3	8	3	3
High	15	1	5	15	1	5	15	1	5

were those selected by F-race as the best performing ones out of the 32 parameter combinations tested.

Table 9.15: Parameters for CDCPSO

Parameter	**Optimisation**	**Total parameters discarded by F-race**	**Bounds**	**Value**
Population size	Fixed	N/A	(0,100]	50
Iterations	Fixed	N/A	$(0,\infty)$	1000
Samples	Fixed	N/A	(0,50)	30
Inertia	F-race	0	(0,1]	0.729844
Social component	F-race	0	(0,3]	1.496180
Cognitive component	F-race	0	(0,3]	1.496180
Total populations	Fixed	N/A	K	8

Table 9.14: F-race Parameters for Tuning the CDCPSO and CDCDynDE

Parameter	**Value**
Iterations	180
Total problems	9
Samples	1 (resulting in 10 per problem)
Minimum number of solutions	2
Parameter configurations	32
Discarding interval	90
Quality measurement	Ray-Turi validity index

Table 9.16: Parameters for CDCDynDE

Parameter	**Optimisation**	**Total parameters discarded by F-race**	**Bounds**	**Value**
Population size	Fixed	N/A	(0,100]	50
Iterations	Fixed	N/A	$(0,\infty)$	1000
Samples	Fixed	N/A	(0,50)	30
% of Brownian individuals	F-race	29	[0,100]	15.625
Exclusion radius	F-race	29	(0,25]	3.90625
Scale parameter	F-race	29	(0,1]	0.53125
Crossover probability	F-race	29	(0,2]	1.6875
Total populations	Fixed	N/A	K	8

9.4.2 Results and Comparison

The average values for the inter-cluster distance, intra-cluster distance and Ray-Turi validity index are reported in Table 9.17. The CDCDynDE obtained the most separated clusters. The CDCPSO, on the other hand, obtained the most compact clusters. The

Ray-Turi validity index demonstrated that the CDCDynDE performed the clustering task more effectively on average, as the algorithm obtained the lowest Ray-Turi validity index value.

Table 9.17: Averages and Standard Deviation for Inter-cluster Distance, Intra-cluster Distance and Ray-Turi Validity for CDCPSO and CDCDynDE Algorithm

Algorithm	**Inter-cluster Distance**	**Intra-cluster Distance**	**Ray-Turi Validity**
CDCPSO	23.394 ± 3.135	**7.230 ± 4.011**	0.989 ± 0.236
CDCDynDE	**30.202** ± 11.341	10.481 ± 8.916	**0.681 ± 0.275**

Figure 9.11 reports the total significant wins and losses obtained by each algorithm. The CDCDynDE obtained significantly more wins than losses when compared to the CDCPSO.

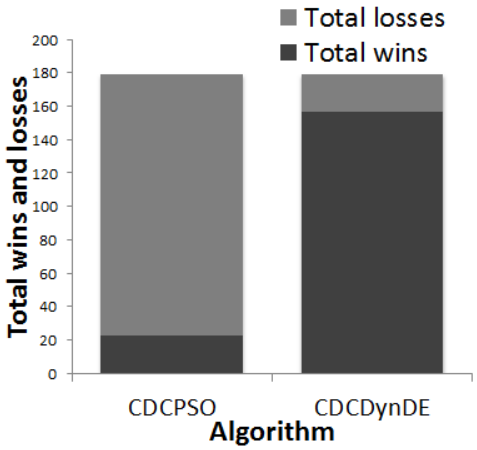


Figure 9.11: Total CDCPSO and CDCDynDE Wins and Losses Per Algorithm Over All Problems

Figure 9.12 illustrates the $Diff$ value of each algorithm on problems of frequencies 1, 2, 3, 4 and 5. The two algorithms were shown both to be insensitive to frequency changes in the previous graphs. This figure illustrates that the CDCDynDE obtained the largest $Diff$ value through all five frequencies, obtaining more wins than losses,

while the CDCPSO remained the worst algorithm through all five frequencies.

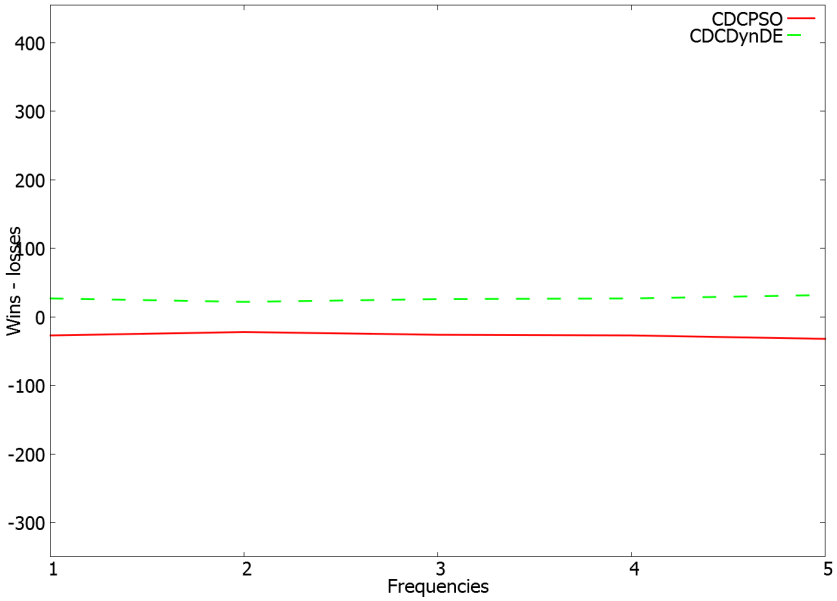


Figure 9.12: $Diff$ for CDCPSO and CDCDynDE Over All Frequencies

Table 9.18 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for the two algorithms over problems of frequencies 1 to 5. The average inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each frequency all correspond to the averages reported in Table 9.17.

Table 9.18: Average CDCPSO and CDCDynDE Results by Frequency

F	Measure	CDCPSO	CDCDynDE
1	Inter-cluster distance	24.761 ± (3.730)	**31.057 ± (10.695)**
	Intra-cluster distance	**8.337 ± (4.217)**	11.414 ± (8.505)
	Ray-Turi validity index	0.991 ± (0.190)	**0.743 ± (0.233)**
	$Diff$	-27	**27**
2	Inter-cluster distance	23.669 ± (3.358)	**31.422 ± (12.065)**

Table 9.18: Average CDCPSO and CDCDynDE Results by Frequency

F	Measure	CDCPSO	CDCDynDE
	Intra-cluster distance	**7.296 ± (4.042)**	11.480 ± (9.421)
	Ray-Turi validity index	0.957 ± (0.206)	**0.708 ± (0.277)**
	Diff	-22	**22**
3	Inter-cluster distance	23.246 ± (2.871)	**31.125 ± (11.171)**
	Intra-cluster distance	**7.116 ± (3.881)**	11.152 ± (8.847)
	Ray-Turi validity index	0.972 ± (0.216)	**0.691 ± (0.272)**
	Diff	-26	**26**
4	Inter-cluster distance	22.792 ± (2.629)	**29.107 ± (10.926)**
	Intra-cluster distance	**6.989 ± (3.989)**	9.696 ± (8.712)
	Ray-Turi validity index	0.996 ± (0.242)	**0.652 ± (0.289)**
	Diff	-27	**27**
5	Inter-cluster distance	22.501 ± (2.339)	**28.297 ± (11.321)**
	Intra-cluster distance	**6.410 ± (3.609)**	8.664 ± (8.616)
	Ray-Turi validity index	1.030 ± (0.302)	**0.614 ± (0.280)**
	Diff	-32	**32**

Figure 9.13 illustrates the performance of each algorithm on problems of severity 1, 2, 3, 4 and 5. Neither of the algorithms was affected by the severity of change. The observations per severity of change correspond to the observations over the frequencies of change.

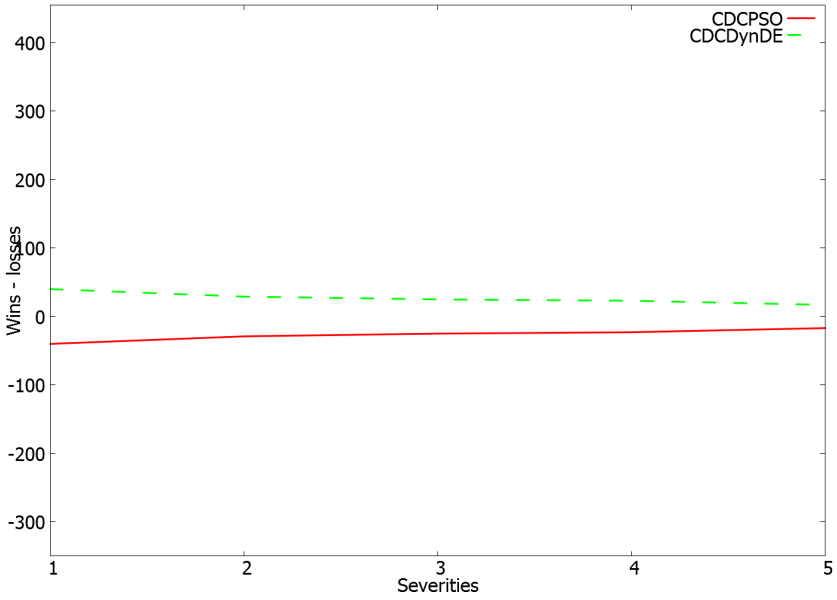


Figure 9.13: $Diff$ for CDCPSO and CDCDynDE Over All Severities

Table 9.19 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each of the algorithms over problems of severities 1 to 5. The observed trends are the same per severity of change as the observed trends per frequency of change.

Table 9.19: Average CDCPSO and CDCDynDE Results by Severity

S	Measure	CDCPSO	CDCDynDE
1	Inter-cluster distance	22.599 ± (3.101)	**23.349 ± (4.828)**
	Intra-cluster distance	**6.453 ± (3.926)**	4.919 ± (4.181)
	Ray-Turi validity index	0.996 ± (0.174)	**0.500 ± (0.245)**
	$Diff$	-40	**40**
2	Inter-cluster distance	23.347 ± (3.234)	**26.837 ± (7.769)**
	Intra-cluster distance	**7.204 ± (4.166)**	8.013 ± (6.356)
	Ray-Turi validity index	0.991 ± (0.234)	**0.650 ± (0.239)**

Table 9.19: Average CDCPSO and CDCDynDE Results by Severity

S	Measure	CDCPSO	CDCDynDE
	Diff	-29	**29**
3	Inter-cluster distance	23.507 ± (3.065)	**32.448 ± (12.204)**
	Intra-cluster distance	**7.141 ± (3.830)**	12.325 ± (9.531)
	Ray-Turi validity index	0.937 ± (0.217)	**0.738 ± (0.264)**
	Diff	-25	**25**
4	Inter-cluster distance	23.597 ± (2.871)	**33.833 ± (12.311)**
	Intra-cluster distance	**7.585 ± (4.016)**	13.303 ± (9.546)
	Ray-Turi validity index	1.018 ± (0.288)	**0.754 ± (0.267)**
	Diff	-23	**23**
5	Inter-cluster distance	23.920 ± (3.203)	**34.542 ± (12.675)**
	Intra-cluster distance	**7.765 ± (3.938)**	13.846 ± (9.807)
	Ray-Turi validity index	1.005 ± (0.244)	**0.764 ± (0.265)**
	Diff	-17	**17**

Figure 9.14 illustrates the performance of each algorithm on 3, 8 and 15 dimensional problems. The CDCDynDE obtained more wins than losses over all dimensions, while the CDCPSO decreased performance in higher dimensions.

Table 9.20 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each DE over 3, 8 and 15 dimensional problems. The observations per dimension are the same as the observations per severity of change.

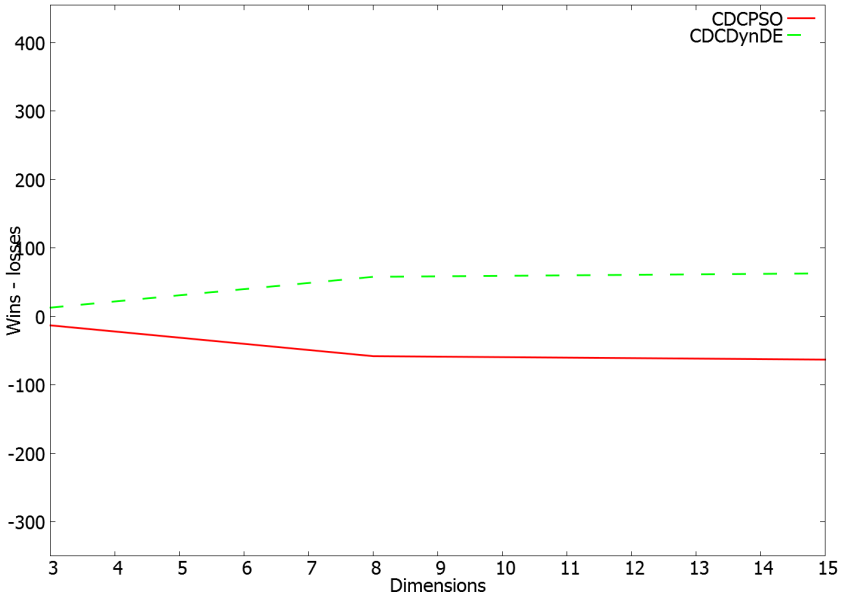


Figure 9.14: $Diff$ for CDCPSO and CDCDynDE Over All Dimensions

Table 9.20: Average CDCPSO and CDCDynDE Results by Dimension

D	Measure	CDCPSO	CDCDynDE
3	Inter-cluster distance	21.694 ± (0.607)	**24.887 ± (3.702)**
	Intra-cluster distance	**3.815 ± (1.753)**	5.569 ± (3.824)
	Ray-Turi validity index	0.920 ± (0.327)	**0.657 ± (0.414)**
	$Diff$	-13	**13**
8	Inter-cluster distance	22.236 ± (2.645)	**29.502 ± (9.298)**
	Intra-cluster distance	**7.450 ± (3.523)**	10.335 ± (7.276)
	Ray-Turi validity index	1.097 ± (0.189)	**0.654 ± (0.210)**
	$Diff$	-58	**58**
15	Inter-cluster distance	26.252 ± (3.098)	**36.216 ± (14.800)**

Table 9.20: Average CDCPSO and CDCDynDE Results by Dimension

D	Measure	CDCPSO	CDCDynDE
	Intra-cluster distance	**10.424 ± (3.265)**	15.540 ± (10.960)
	Ray-Turi validity index	0.951 ± (0.080)	**0.733 ± (0.084)**
	$Diff$	-63	**63**

Figure 9.15 illustrates the total wins and losses for each algorithm per migration type. The CDCDynDE obtained significantly more wins than losses for pattern migration problems. CDCPSO demonstrated a small improvement for centroid migration problems, but remained with more losses than wins. Lastly, the CDCPSO significantly improved performance when applied to cluster migration problems, but still remained the worst algorithm.

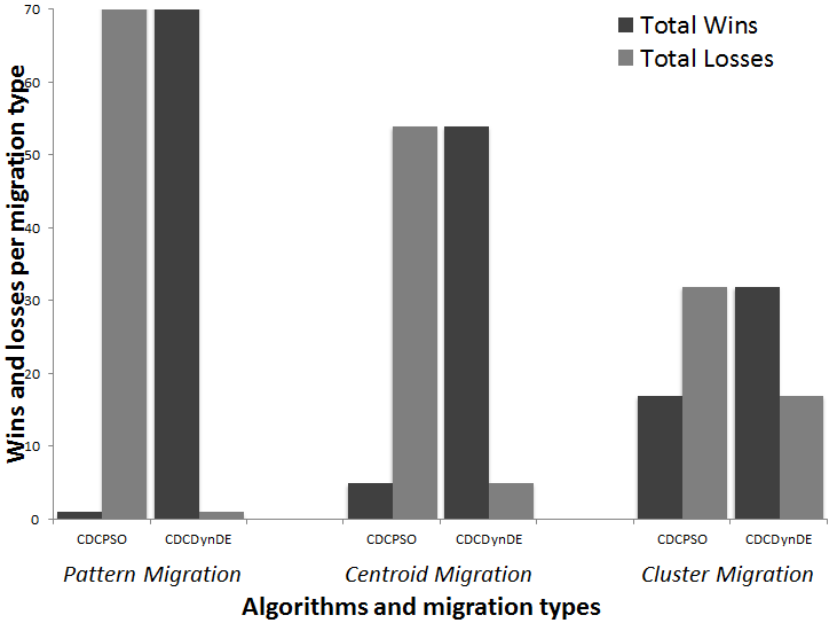


Figure 9.15: $Diff$ for CDCPSO and CDCDynDE Over All Migration Types

Table 9.21 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each DE over pattern migration, cluster migration and centroid migration

problems. The average results per migration type also correspond to the average results per frequency, severity and dimension.

Table 9.21: Average CDCPSO and CDCDynDE Results by Migration Type

M	Measure	CDCPSO	CDCDynDE
Pattern	Inter-cluster distance	21.778 ± (1.587)	**22.237 ± (1.949)**
	Intra-cluster distance	5.037 ± (2.307)	**3.840 ± (2.182)**
	Ray-Turi validity index	0.895 ± (0.157)	**0.454 ± (0.221)**
	Diff	-70	**70**
Centroid	Inter-cluster distance	22.720 ± (2.155)	**32.345 ± (11.825)**
	Intra-cluster distance	**5.732 ± (2.565)**	12.410 ± (9.019)
	Ray-Turi validity index	0.876 ± (0.149)	**0.769 ± (0.223)**
	Diff	-49	**49**
Cluster	Inter-cluster distance	25.684 ± (3.727)	**36.023 ± (11.771)**
	Intra-cluster distance	**10.920 ± (3.935)**	15.194 ± (9.017)
	Ray-Turi validity index	1.197 ± (0.236)	**0.821 ± (0.220)**
	Diff	-15	**15**

The results show that the CDCDynDE is more efficient than the CDCPSO in clustering temporal data. For this reason, the CDCDynDE was chosen as the algorithm to be adapted in order to determine the optimal number of clusters dynamically. The adaptation of this algorithm resulted in the KCDCDynDE described in Chapter 8.

9.5 K-independent Cooperative Data Clustering Differential Evolution Versus Local Network Neighbourhood Artificial Immune System

This section compares the results of the KCDCDynDE algorithm to the results of the LNNAIS algorithm. The two algorithms are applied to the artificially generated datasets used in the previous sections. The KCDCDynDE algorithm was compared against Graaff and Engelbrecht's [46] LNNAIS algorithm because the same datasets and experimental procedure were used for the results of both algorithms.

9.5.1 Experimental Set-up

Graaff and Engelbrecht [46] optimised the parameters of the LNNAIS algorithm empirically for each of the 225 problems. These parameters are summarised in Appendix A. The parameters of the KCDCDynDE were also determined empirically for each of the 225 problems and are summarised in Appendix B.

9.5.2 Results and Comparison

Table 9.22 reports the average values for the inter-cluster distance, intra-cluster distance and Ray-Turi validity index. The KCDCDynDE obtained the highest inter-cluster distance between the two algorithms, while the LNNAIS obtained the lowest intra-cluster distance. The KCDCDynDE performed the clustering task more effectively on average, as the algorithm obtained the lowest Ray-Turi validity index value.

Figure 9.16 reports the total significant wins and losses obtained by each algorithm. The KCDCDynDE obtained more total wins than losses when compared to the LNNAIS.

Table 9.22: Averages and Standard Deviation for Inter-cluster Distance, Intra-cluster Distance and Ray-Turi Validity for LNNAIS and KCDCDynDE Algorithm

Algorithm	**Inter-cluster Distance**	**Intra-cluster Distance**	**Ray-Turi Validity**
LNNAIS	20.219 ± 1.137	**4.986 ± 2.282**	0.789 ± 0.307
KCDCDynDE	**22.969 ± 3.936**	6.257 ± 4.385	**0.598 ± 0.239**

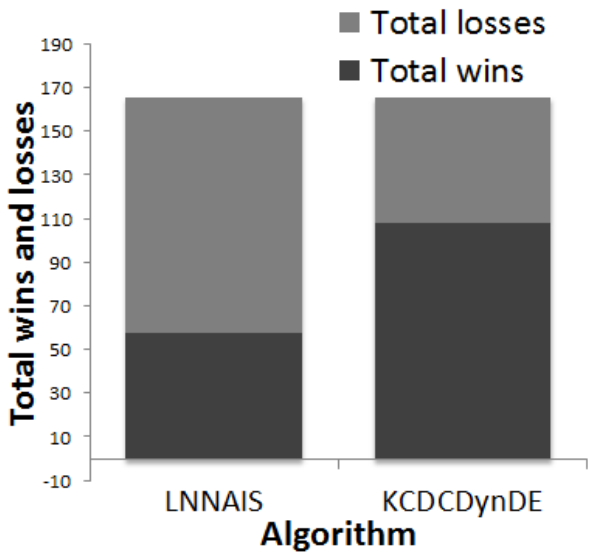


Figure 9.16: Total LNNAIS and KCDCDynDE Wins and Losses Per Algorithm Over All Problems

Figure 9.17 illustrates the $Diff$ value of each algorithm on problems of frequencies 1, 2, 3, 4 and 5. The KCDCDynDE obtained more wins than losses for lower frequency values (more frequent changes). The LNNAIS, however, improved performance as the frequency value increased (less changes occur in the dataset), performing better than the KCDCDynDE for problems of frequency four.

Figure 9.18 illustrates the difference between the expected K and actual K for each of the two algorithms across the various frequencies. The difference was calculated for each time-step before a change occurs and averaged over the total time-steps before a change occurs and the total samples.

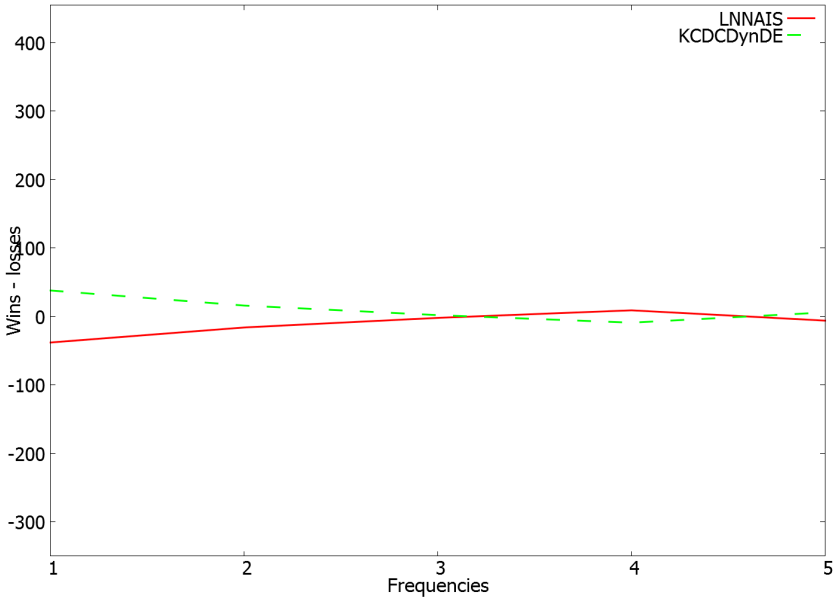


Figure 9.17: $Diff$ for LNNAIS and KCDCDynDE Over All Frequencies

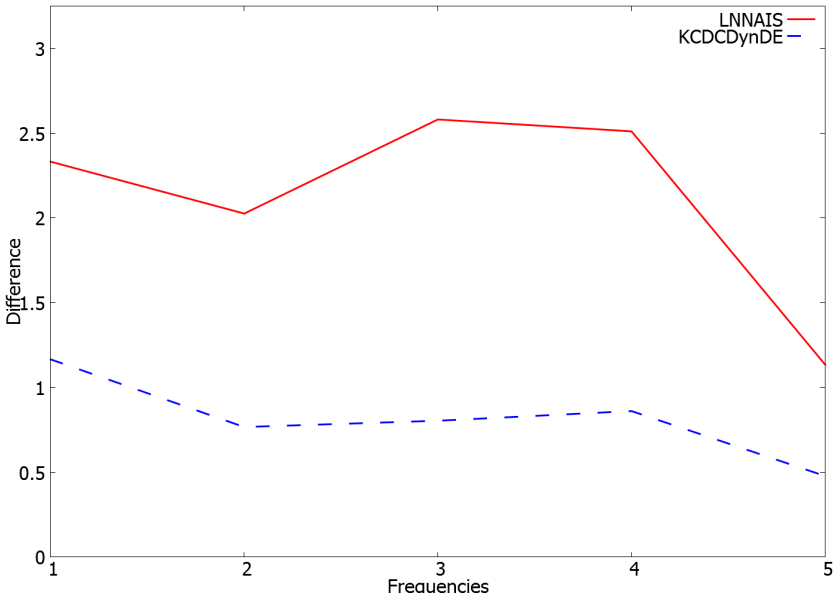


Figure 9.18: Difference Between Expected K and Resulting K for Each Frequency

Figure 9.18 demonstrates that the KCDCDynDE obtained the closest K to the expected

value over all frequencies. Both algorithms obtained a more accurate K for problems with less changes (frequency of 5). The least accurate K obtained by the LNNAIS was for problems of frequency 3, and for problems of frequency 1 for the KCDCDynDE.

Table 9.23 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each algorithm over problems of frequencies 1 to 5. The resulting trends correspond to the trends summarised by the averages in Table 9.22;

Table 9.23: Average LNNAIS and KCDCDynDE Results by Frequency

F	Measure	LNNAIS	KCDCDynDE
1	K	4.795 ± (1.542)	6.227 ± (1.087)
	Inter-cluster distance	19.615 ± (1.488)	**22.278 ± (3.023)**
	Intra-cluster distance	**4.761 ± (1.846)**	6.009 ± (3.561)
	Ray-Turi validity index	0.974 ± (0.198)	**0.569 ± (0.189)**
	$Diff$	-38	**38**
2	K	4.889 ± (1.624)	6.795 ± (1.009)
	Inter-cluster distance	20.210 ± (0.958)	**23.696 ± (4.560)**
	Intra-cluster distance	**4.919 ± (2.201)**	6.725 ± (4.849)
	Ray-Turi validity index	0.850 ± (0.234)	**0.602 ± (0.220)**
	$Diff$	-16	**16**
3	K	4.805 ± (1.640)	7.025 ± (0.758)
	Inter-cluster distance	20.353 ± (0.820)	**23.462 ± (4.017)**
	Intra-cluster distance	**4.917 ± (2.259)**	6.759 ± (4.710)
	Ray-Turi validity index	0.718 ± (0.288)	**0.637 ± (0.265)**
	$Diff$	-2	**2**
4	K	4.869 ± (1.705)	7.179 ± (0.895)
	Inter-cluster distance	20.417 ± (1.014)	**22.844 ± (3.649)**
	Intra-cluster distance	**4.943 ± (2.238)**	6.457 ± (4.536)

Table 9.23: Average LNNAIS and KCDCDynDE Results by Frequency

F	Measure	LNNAIS	KCDCDynDE
	Ray-Turi validity index	0.669 ± (0.287)	**0.493 ± (0.268)**
	$Diff$	**9**	-9
5	K	5.093 ± (1.829)	7.363 ± (0.820)
	Inter-cluster distance	20.496 ± (0.873)	**22.682 ± (3.932)**
	Intra-cluster distance	**5.006 ± (2.618)**	5.326 ± (3.969)
	Ray-Turi validity index	0.682 ± (0.330)	**0.426 ± (0.228)**
	$Diff$	-6	**6**

Figure 9.19 illustrates the performance of each algorithm on problems of severity 1, 2, 3, 4 and 5. The KCDCDynDE obtained a positive $Diff$ value for four of the five severities, obtaining more wins in lower severities. The LNNAIS obtained more losses than wins, but improved performance as severities increased.

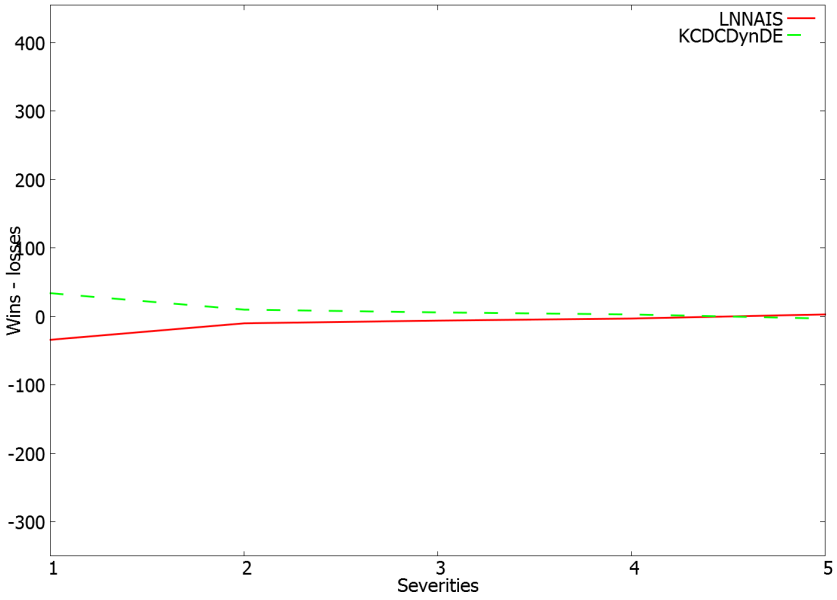


Figure 9.19: $Diff$ for LNNAIS and KCDCDynDE Over All Severities

Figure 9.20 illustrates the difference between the expected K and actual K for each of the two algorithms across the various severities. The overall observations are the same as across the various frequencies, where the KCDCDynDE obtained the closest K to the expected value over all severities. The KCDCDynDE is more accurate on lower severities of change than on higher ones, while the LNNAIS is more accurate on problems of severity of 5 than lower severities.

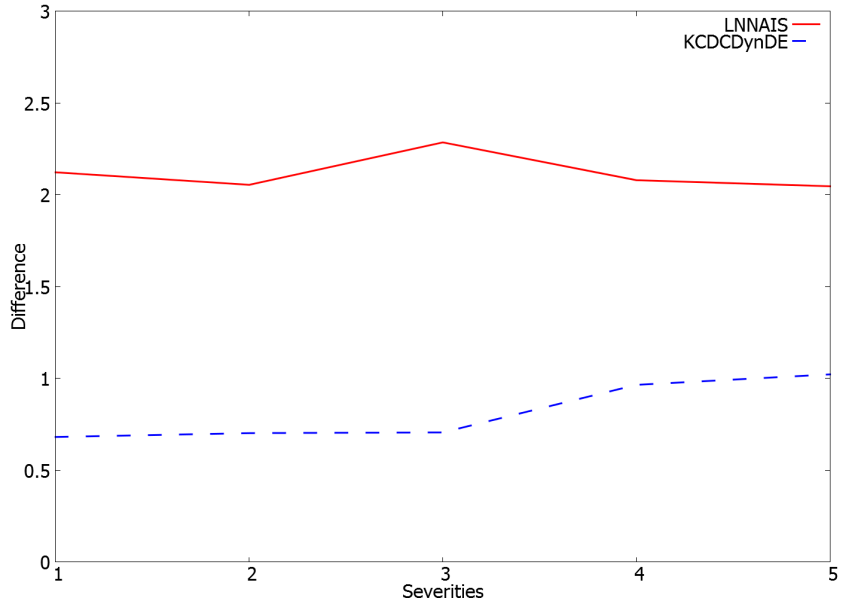


Figure 9.20: Difference Between Expected K and Resulting K for Each Severity

Table 9.24 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each DE over problems of severities 1 to 5. The average observations are the same as those per frequency of change.

Table 9.24: Average LNNAIS and KCDCDynDE Results by Severity

S	Measure	LNNAIS	KCDCDynDE
1	K	5.438 ± (1.960)	7.328 ± (0.595)
	Inter-cluster distance	20.357 ± (0.917)	**20.838 ± (0.726)**
	Intra-cluster distance	5.054 ± (2.438)	**3.468 ± (1.480)**
	Ray-Turi validity index	0.776 ± (0.304)	**0.458 ± (0.175)**

Table 9.24: Average LNNAIS and KCDCDynDE Results by Severity

S	Measure	LNNAIS	KCDCDynDE
	Diff	-34	**34**
2	K	5.336 ± (1.688)	7.027 ± (0.777)
	Inter-cluster distance	20.077 ± (1.308)	**21.693 ± (2.006)**
	Intra-cluster distance	**4.821 ± (2.246)**	5.216 ± (2.691)
	Ray-Turi validity index	0.787 ± (0.304)	**0.578 ± (0.213)**
	Diff	-10	**10**
3	K	4.730 ± (1.540)	6.933 ± (0.804)
	Inter-cluster distance	20.106 ± (1.228)	**24.662 ± (5.268)**
	Intra-cluster distance	**4.841 ± (2.056)**	7.536 ± (5.508)
	Ray-Turi validity index	0.783 ± (0.277)	**0.640 ± (0.254)**
	Diff	-6	**6**
4	K	4.638 ± (1.451)	6.823 ± (1.113)
	Inter-cluster distance	20.213 ± (0.880)	**24.005 ± (4.322)**
	Intra-cluster distance	**4.804 ± (2.103)**	7.444 ± (4.633)
	Ray-Turi validity index	0.813 ± (0.323)	**0.503 ± (0.235)**
	Diff	-3	**3**
5	K	4.308 ± (1.386)	6.479 ± (1.340)
	Inter-cluster distance	20.339 ± (1.092)	**23.765 ± (3.804)**
	Intra-cluster distance	**5.026 ± (2.357)**	7.612 ± (4.600)
	Ray-Turi validity index	0.734 ± (0.260)	**0.523 ± (0.261)**
	Diff	**3**	-3

Figure 9.21 illustrates the performance of each algorithm on 3, 8 and 15 dimensional problems. The KCDCDynDE obtained more wins than losses over 8 and 15 dimen-

sional problems and half of the three dimensional problems. The LNNAIS algorithm's performance, on the other hand, decreased as dimensions increased.

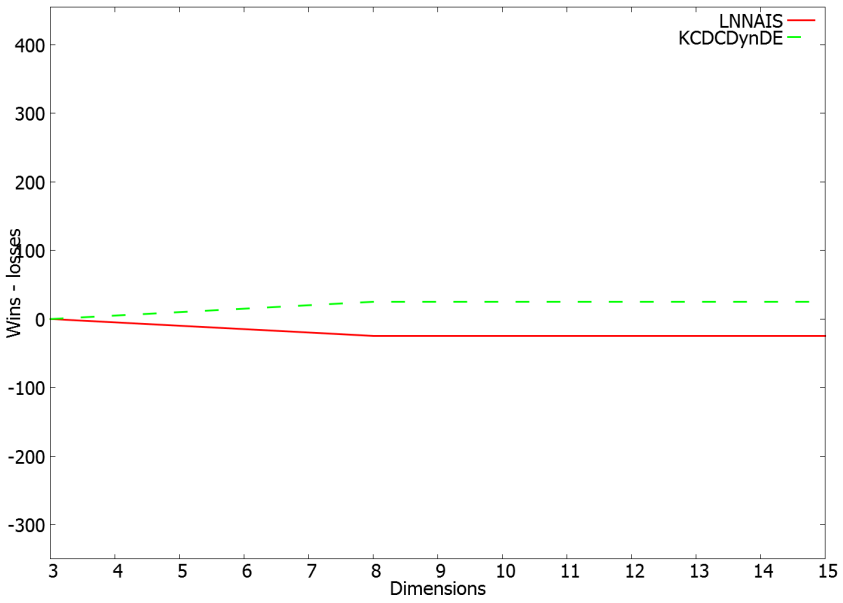


Figure 9.21: $Diff$ for LNNAIS and KCDCDynDE Over All Dimensions

Figure 9.22 illustrates the difference between the expected K and actual K for each of the two algorithms across the various dimensions. The LNNAIS obtained a slightly more accurate K than the KCDCDynDE in three-dimensional problems. However, the KCDCDynDE obtains a much more accurate K for problems with dimensions eight and 15, where the LNNAIS's accuracy significantly lowers.

Table 9.25 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each algorithm over 3, 8 and 15 dimensional problems. The average results per dimension correspond to the average results per severity and frequency.

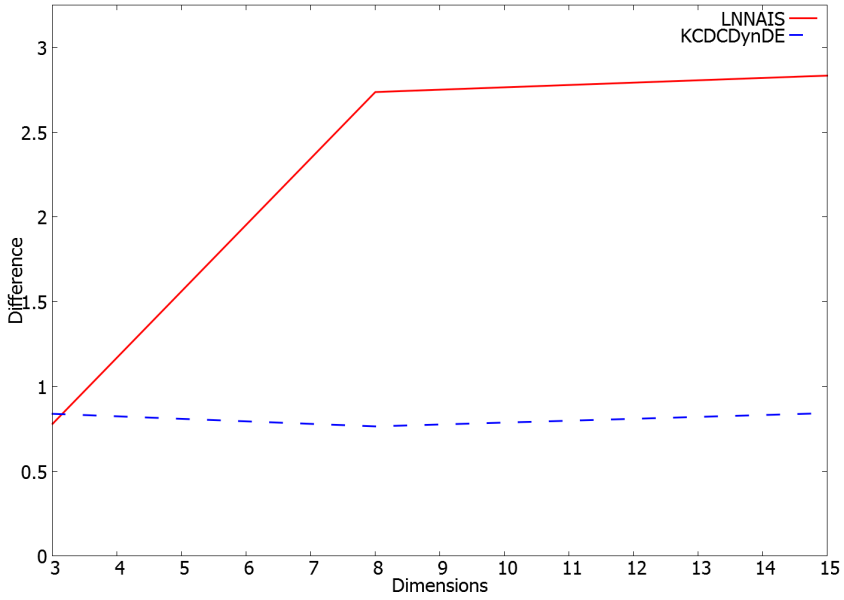


Figure 9.22: Difference Between Expected K and Resulting K for Each Dimension

Table 9.25: Average LNNAIS and KCDCDynDE Results by Dimension

D	Measure	LNNAIS	KCDCDynDE
3	K	6.457 ± (1.463)	7.254 ± (0.814)
	Inter-cluster distance	20.956 ± (1.157)	**22.192 ± (1.447)**
	Intra-cluster distance	**2.280 ± (0.723)**	3.674 ± (2.121)
	Ray-Turi validity index	0.511 ± (0.297)	**0.290 ± (0.243)**
	$Diff$	0	0
8	K	4.206 ± (1.215)	6.721 ± (0.947)
	Inter-cluster distance	19.899 ± (0.794)	**21.718 ± (2.535)**
	Intra-cluster distance	**5.416 ± (1.184)**	5.898 ± (3.354)
	Ray-Turi validity index	0.864 ± (0.187)	**0.411 ± (0.245)**

Table 9.25: Average LNNAIS and KCDCDynDE Results by Dimension

D	Measure	LNNAIS	KCDCDynDE
	$Diff$	-25	**25**
15	K	4.007 ± (1.044)	6.778 ± (1.131)
	Inter-cluster distance	19.800 ± (0.931)	**25.067 ± (5.534)**
	Intra-cluster distance	**7.032 ± (1.245)**	9.194 ± (5.149)
	Ray-Turi validity index	0.960 ± (0.164)	**0.722 ± (0.136)**
	$Diff$	-25	**25**

Figure 9.23 illustrates the total wins and losses for each algorithm per migration type. The KCDCDynDE obtained significantly more wins than losses for pattern migration and cluster migration problems. LNNAIS, however, outperformed the KCDCDynDE on centroid migration problems.

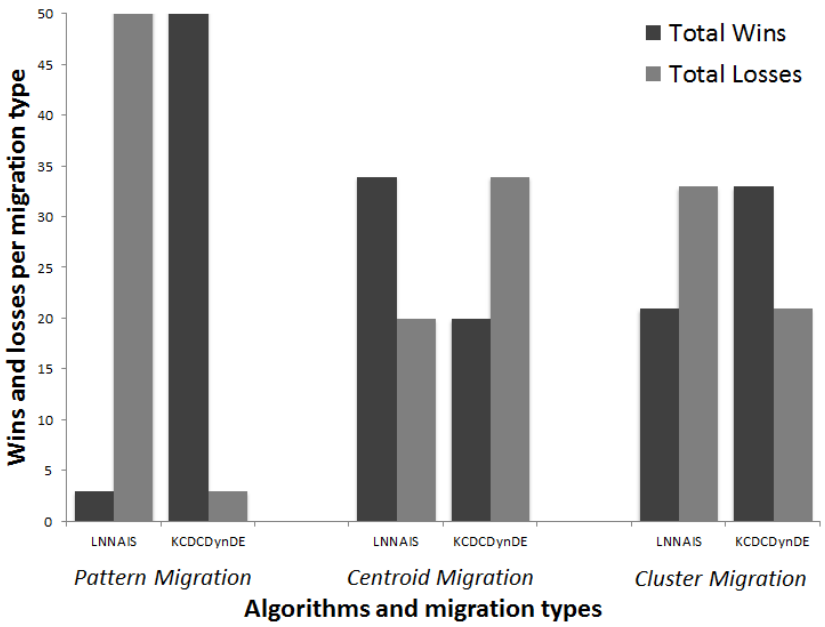


Figure 9.23: $Diff$ for LNNAIS and KCDCDynDE Over All Migration Types

Figure 9.24 illustrates the difference between the expected K and actual K for each of the two algorithms for each migration pattern. The KCDCDynDE obtained the best results over all migration types, corresponding to the results over all severities and frequencies. The most accurate K obtained by the KCDCDynDE was for pattern migration problems, and cluster migration problems for the LNNAIS.

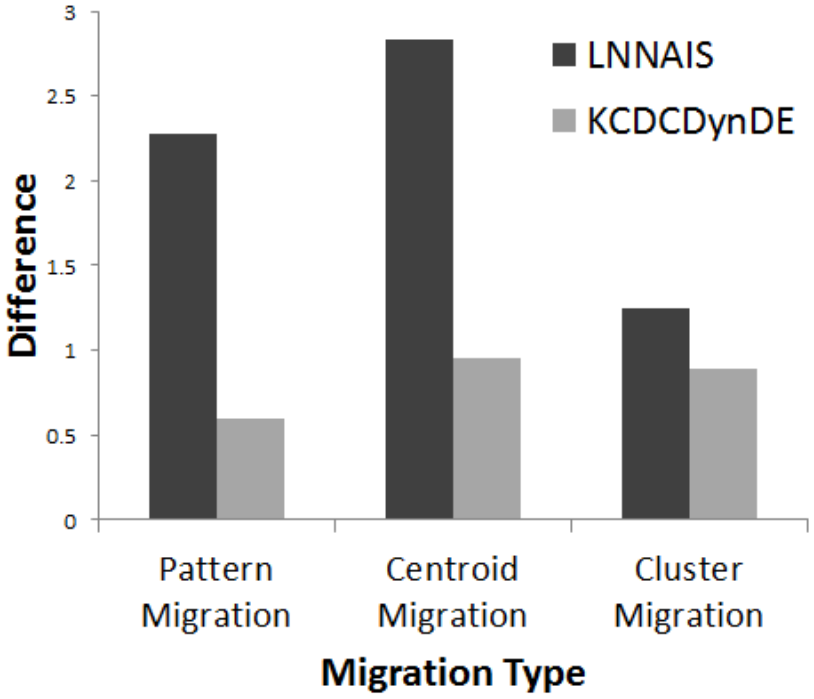


Figure 9.24: Difference Between Expected K and Resulting K for Each Migration Type

Table 9.26 summarises the inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each algorithm over the different migration types. The KCDCDynDE resulted in the highest average inter-cluster distance as well as the lowest average Ray-Turi validity index over all migration types. The LNNAIS obtained the lowest average intra-cluster distance for centroid and cluster migration problems.

To summarise, the Ray-Turi validity index and the wins and losses approach demonstrated that the KCDCDynDE algorithm performed the clustering task better than the LNNAIS algorithm for more frequently changing data (lower frequency values), high-dimensional data, and pattern and cluster migration problems. The results also

Table 9.26: Average LNNAIS and KCDCDynDE Results by Migration Type

M	Measure	LNNAIS	KCDCDynDE
Pattern	K	5.325 ± (2.039)	7.439 ± (0.458)
	Inter-cluster distance	20.541 ± (0.723)	**21.170 ± (0.950)**
	Intra-cluster distance	4.946 ± (2.436)	**3.363 ± (1.366)**
	Ray-Turi validity index	0.783 ± (0.323)	**0.436 ± (0.178)**
	Diff	-52	**52**
Centroid	K	4.646 ± (1.530)	6.371 ± (1.214)
	Inter-cluster distance	20.667 ± (0.670)	**23.482 ± (3.770)**
	Intra-cluster distance	**4.307 ± (2.075)**	6.403 ± (3.834)
	Ray-Turi validity index	0.732 ± (0.304)	**0.375 ± (0.208)**
	Diff	**14**	-14
Cluster	K	4.699 ± (1.275)	6.943 ± (0.869)
	Inter-cluster distance	19.447 ± (1.336)	**24.325 ± (5.026)**
	Intra-cluster distance	**5.474 ± (2.056)**	9.000 ± (5.020)
	Ray-Turi validity index	0.821 ± (0.248)	**0.517 ± (0.283)**
	Diff	-12	**12**

demonstrated that the LNNAIS algorithm was more appropriate for centroid migration problems and higher severity datasets.

9.6 Summary

This chapter summarised the empirical results of applying the algorithms proposed in this research to 225 artificial temporal clustering datasets. First the results of the proposed PSOs were summarised, where the CDCPSO performed the clustering task best. This was followed by the results of the proposed DEs, where the CDCDynDE performed the clustering task best. The CDCPSO and the CDCDynDE were then compared in order to determine which of the two algorithms performed better. The CDCDynDE was selected due to its superior performance and the absence of the outdated memory in DE algorithms when applied to dynamic environments. The CDCDynDE was then adapted to dynamically determine the total number of clusters.

The results of the adapted CDCDynDE, namely the KCDCDynDE, were reported and compared to the results of the LNNAIS. It was demonstrated that the KCDCDynDE performed well on frequently changing data, high-dimensional data and pattern and cluster migration datasets. The LNNAIS, on the other hand, was shown to perform better on data with high severities of change and centroid migration datasets.

Chapter 10

Conclusions

This chapter summarises the conclusions of the research presented in this thesis. Section 10.1 summarises the conclusions derived at through the research performed, while Section 10.2 summarises possible future work that can be done on the topic.

10.1 Summary of Conclusions

This research aimed to develop particle swarm optimisation (PSO) and differential evolution (DE) data clustering algorithms that can be used to cluster temporal datasets. Two limitations that the PSO algorithm faces when applied to temporal environments were identified. These limitations are the loss of diversity and outdated memory. Of these limitations, DE algorithms only face the loss of diversity. Additionally, various previously used approaches of dealing with the above-mentioned limitations were described. These approaches include re-initialisation, anti-convergence and repulsion. Computational intelligence (CI) approaches previously used to cluster static and temporal data were also summarised along with techniques used to determine the total number of clusters in a dataset dynamically.

Various existing PSO algorithms were selected and adapted to perform the task of clus-

tering temporal data. The performance of these algorithms was compared against one another using a wins and losses approach. The average results were also discussed. It was demonstrated that the proposed cooperative multi-swarm data clustering particle swarm optimisation (CMDCPSO) and elitist cooperative multi-swarm data clustering particle swarm optimisation (eCMDCPSO) algorithms did not perform the clustering tasks well. The anti-convergence component of the CMDCPSO and eCMDCPSO was mentioned as a possible cause for the failure of the two algorithms, where the final solution can result in an obsolete centroid. This obsolete centroid is caused by the anti-convergence technique re-initialising a population if all populations converge. When a population is re-initialised, it becomes an explorer population that searches the workspace independent of the context particle. The re-initialised population is replaced with the explorer population and the centroid that the new population contributes to the context particle can be obsolete or may already exist in the context. The PSO that performed the data clustering tasks best was shown to be the cooperative data clustering particle swarm optimisation (CDCPSO), followed by the multi-swarm data clustering particle swarm optimisation (MDCPSO).

In addition to the proposed PSOs, a variety of existing DE algorithms was selected and adapted to perform the task of clustering temporal data. It was demonstrated that, among the DEs, the cooperative dynamic data clustering differential evolution (CDCDynDE) performed the clustering tasks most effectively. Additionally, the dynamic data clustering differential evolution (DCDynDE) was shown to be sensitive to increases in the dimensionality of the problem.

The performance of the best PSO and best DE was compared, namely the CDCPSO and CDCDynDE. The CDCDynDE performed the clustering task best regardless of changes in frequency, severity, dimension and migration type. The most notable behaviour of the algorithms was the increased performance of the CDCDynDE in higher dimensions and the improvement in the performance of the CDCPSO in cluster migration problems. The comparison of the CDCDynDE and CDCPSO was aimed at selecting the most appropriate algorithm to adapt in order to dynamically determine the optimal number of clusters. The CDCDynDE was chosen due to its superior performance. Another

advantage of selecting the CDCDynDE is that DEs do not suffer from the problem of outdated memory that PSOs experience, and therefore require no mechanism to deal with the possibility of outdated memory.

The k-independent cooperative data clustering differential evolution (KCDCDynDE) algorithm was applied to the 225 problems and compared to the local network neighbourhood artificial immune system (LNNAIS). The two algorithms were compared to one another due to their ability to determine the total number of clusters dynamically. The LNNAIS demonstrated superior performance when changes in the dataset occurred less frequently (larger frequency values), while the KCDCDynDE performed better when more changes in the dataset occurred. The KCDCDynDE also demonstrated superior performance for lower severities of change and a similar performance to the LNNAIS as severities increased. Although both algorithms demonstrated the same performance for three-dimensional problems, the KCDCDynDE significantly outperformed the LNNAIS in higher dimensions. Lastly, the KCDCDynDE obtained more wins than the LNNAIS for pattern migration and cluster migration problems, but was the worst algorithm for centroid migration problems.

The results of comparing the KCDCDynDE to the LNNAIS imply that the KCDCDynDE is an appropriate algorithm to cluster frequently changing data, higher-dimensional data and pattern and cluster migration datasets. The LNNAIS, however, appears to be more appropriate for data with higher severities as well as for centroid migration datasets.

In summary, this research has contributed a variety of algorithms that can be used for clustering temporal and static data. The main contribution is the KCDCDynDE algorithm, which determines the number of clusters dynamically and clusters temporal data. Results demonstrating the performance of all the 12 described algorithms are made available in this document and can be re-used in further research.

10.2 Future Work

This research focused on the use of PSO and DE algorithms to cluster temporal data. Future work that can emerge from this research includes:

- A scalability study of the described algorithms.
- Analysing the impact of using fitness functions other than the quantization error.
- Implementing a multi-objective clustering approach where the inter-cluster distance is minimised and the intra-cluster distance is maximised.
- Implementing a multi-objective clustering approach that combines a variety of validity indices.
- An analysis of the effect of regenerating the context individual when the best individual of the population is updated. This is applicable to the KCDCDynDE, CDCDynDE, CMDCPSO and eCMDCPSO.
- A comparison of the performance of the KCDCDynDE against other types of algorithms, such as the Kohonen self-organisation map neural network (KSOM-NN) [120], a clustering self-organising map (SOM) [43], an artificial neural network (ANN) approach to clustering time-series data [127], and a combination of SOM and autoregressive integrated moving average (ARIMA) time-series models [119].
- An analysis of the impact that each parameter has on the proposed clustering algorithms, and the development of self-adaptive versions of these algorithms.
- An implementation of the CMDCPSO and eCMDCPSO without the anti-convergence component in order to determine whether the anti-convergence component caused the CMDCPSO and eCMDCPSO to fail at the clustering task.
- An adaptation of the KCDCDynDE to also remove a population if the centroid that the population contributed to the context individual is assigned no data patterns.

Bibliography

[1] A. Abraham, S. Das, and S. Roy. Swarm intelligence algorithms for data clustering. *Soft Computing for Knowledge Discovery and Data Mining*, (4):279–313, 2008.

[2] M. B. Al-Zoubi and M. Al Rawi. An efficient approach for computing silhouette coefficients. *Journal of Computer Science*, 4(3):252–255, 2008.

[3] C. M. Antunes and A. L. Oliveira. Temporal data mining: an overview. pages 1–13, August 2001.

[4] A. Atyabi, S. Phon-Amnuaisuk, and C. K. Ho. Navigating a robotic swarm in an uncharted 2d landscape. *Applied Soft Computing*, 10(1):149 – 169, 2010.

[5] K. Bache and M. Lichman. UCI machine learning repository [online]. Available: http://archive.ics.uci.edu/ml (Accessed: 15 January 2014) University of California, Irvine, School of Information and Computer Sciences, 2013.

[6] D. Birant and A. Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[7] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proc. Genetic and Evolutionary Computation Conference*, volume 2, pages 11–18, 2002.

[8] C. M. Bishop. *Neural networks for pattern recognition*. Clarendon Press, 1995.

[9] T. Blackwell. Particle swarm optimization in dynamic environments. *Evolutionary*

Computation in Dynamic and Uncertain Environments, 51:29–49, 2007.

[10] T. Blackwell and J Branke. Multi-swarm optimization in dynamic environments. *Applications of Evolutionary Computing*, 3005:489–500, 2004.

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

[12] T. M. Blackwell and P. J. Bentley. Dynamic search with charged swarms. In *Proc. Genetic and Evolutionary Computation Conference*, volume 2, pages 19–26, 2002.

[13] Yevgeniy Bodyanskiy. Computational intelligence techniques for data analysis. In *Proc. Leipziger Informatik-Tage*, pages 15–36, 2005.

[14] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, 1999.

[15] J. Branke, T. Kaußler, C. Schmidt, and Hartmut. A multi-population approach to dynamic optimization problems. In *Proc. In Adaptive Computing in Design and Manufacturing*, pages 299–308, 2000.

[16] J. Brownlee. A review of the clonal selection theory of acquired immunity. *The Journal of the American Medical Association*, 105(6):973–974, 1960.

[17] C. Chen and F. Ye. Particle swarm optimization algorithm and its application to clustering analysis. In *Proc. IEEE International Conference on Networking, Sensing and Control*, volume 2, pages 789–794, 2004.

[18] T. Cheng and J. Wang. Integrated spatio-temporal data mining for forest fire prediction. *Transactions in Geographical Information Systems*, 12(5):591–611, 2008.

[19] L. Chittaro, C. Combi, and G. Trapasso. Data mining on temporal data: a visual approach and its clinical application to hemodialysis. *Journal of Visual Languages and Computing*, 14(6):591–620, 2003.

[20] T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris. A robust and scalable clustering algorithm for mixed type attributes in large database environment. In *Proc. Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 263–268, 2001.

[21] F. E. Clements. Use of cluster analysis with anthropological data. *American Anthropologist*, 56(2):180–199, 1954.

[22] T. Cloete, A. P. Engelbrecht, and G. Pampara. Cilib: A collaborative framework for computational intelligence algorithms-part ii. In *Proc. IEEE International Joint Conference on Neural Networks*, pages 1764–1773, 2008.

[23] S. C. M. Cohen and L. N. de Castro. Data clustering with particle swarms. In *Proc. IEEE Congress on Evolutionary Computation*, pages 1792–1798, 2006.

[24] C. Darwin. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*. John Murray, 1 edition, 1859.

[25] S. Das, A. Abraham, K. Chakraborty, and A. Konar. Differential evolution using a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526 – 553, June 2009.

[26] S. Das, A. Abraham, and A. Konar. Automatic clustering using an improved differential evolution algorithm. *IEEE Transactions on Systems Man and Cybernetics Part A Systems and Humans*, 38(1):218–237, 2008.

[27] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, 1979.

[28] F. Van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, November 2001.

[29] K. Do-Jong. A novel validity index for determination of the optimal number of clusters. *IEICE Transactions on Information and Systems*, 84(2):281–285, 2001.

[30] M. C. du Plessis and A. P. Engelbrecht. Improved differential evolution for dynamic

optimization problems. In *Proc. IEEE Congress on Evolutionary Computation IEEE World Congress on Computational Intelligence*, pages 229–234, 2008.

[31] J. G. O. L. Duhain and A. P. Engelbrecht. Towards a more complete classification system for dynamically changing environments. In *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2012.

[32] J. C. Dunn. Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[33] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proc. Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[34] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proc. Congress on Evolutionary Computation*, volume 1, pages 84–88, 2000.

[35] R. C. Eberhart and Y. Shi. *Computational intelligence: concepts to implementations*, chapter 1. Kaufmann Publishers, 2007.

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

[37] A. P. Engelbrecht. *Computational Intelligence: An Introduction*, chapter 1. Wiley Publishing, 2007.

[38] J. D. Farmer, N. H. Packard, and A. S. Perelson. The immune system, adaptation, and machine learning. *Physica 22D*, 2(3):187–204, November 1986.

[39] L. C. M. Félix, S. O. Rezende, M. C. Monard, and C. W. Caulkis. Transforming a regression problem into a classification problem using hybrid discretization. *Computació andn y Sistemas*, 4(1):44–52, 2000.

[40] G. Fogel and D. Corne. *Evolutionary computation in bioinformatics*. Morgan

Kauffman, 2003.

[41] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 202–212, 1994.

[42] H. Frigui and R. Krishnapuram. Clustering by competitive agglomeration. *Pattern recognition*, 30(7):1109–1119, 1997.

[43] T-c Fu, F-l Chung, V. Ng, and R. Luk. Pattern discovery from stock time series using self-organizing maps. In *Proc. Workshop Notes of KDD2001 Workshop on Temporal Data Mining*, pages 26–29, 2001.

[44] K. Georgieva and A. P. Engelbrecht. A cooperative multi-population approach to clustering temporal data. In *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pages 1983–1991, 2013.

[45] K. Georgieva and A. P. Engelbrecht. Dynamic differential evolution algorithm for clustering temporal data. In Ivan Lirkov, Svetozar Margenov, and Jerzy Waśniewski, editors, *Large-Scale Scientific Computing*, Lecture Notes in Computer Science, pages 240–247. Springer Berlin Heidelberg, 2014.

[46] A. Graaff. *A Local Network Neighbourhood Artificial Immune System*. PhD thesis, University of Preotria, June 2011.

[47] A. J. Graaff and A. P. Engelbrecht. A local network neighbourhood artificial immune system for data clustering. In *Proc. IEEE Congress on Evolutionary Computation*, pages 260–267, 2007.

[48] R. R Gutell and R. K Jansen. *Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion*. PhD thesis, The University of Texas at Austin, 2006.

[49] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001.

[50] J. Handl and J. Knowles. Multi-objective clustering and cluster validation. In Yaochu Jin, editor, *Multi-Objective Machine Learning*, volume 16 of *Studies in Computational Intelligence*, pages 21–47. Springer Berlin Heidelberg, 2006.

[51] S. A. Hanuman, V. A. Babu, A. Govardhan, and S. C. Satapathy. Data clustering using almost parameter free differential evolution technique. *International Journal of Computer Applications*, 8(13):1–7, October 2010.

[52] M. Helbig and A. P. Engelbrecht. Issues with performance measures for dynamic multi-objective optimisation. In *Proc. IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments*, pages 17–24, 2013.

[53] J. Herrero, A. Valencia, and J. Dopazo. A hierarchical unsupervised growing neural network for clustering gene expression patterns. *Bioinformatics*, 17(2):126–136, 2001.

[54] X. Hou, L. Han, M. Gao, X. Bi, and M. Zhu. Application of spatio-temporal data mining and knowledge discovery for detection of vegetation degradation. In *Proc. Seventh International Conference on Fuzzy Systems and Knowledge Discovery*, volume 5, pages 2124–2128, 2010.

[55] Z. Huang. Clustering large data sets with mixed numeric and categorical values. In *Proc. The First PacificAsia Conference on Knowledge Discovery and Data Mining*, pages 21–34, 1997.

[56] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[57] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[58] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.

[59] A. E. Johnson and A. S. Johnson. K-means and temporal variability in kansas city

hopewell ceramics. *American Antiquity*, 40(3):283–295, 1975.

[60] A. Juhász. Managing temporal data in military historical gis. In *Proc. 31th Symposium of the European Association of Remote Sensing Laboratories*, pages 978–980, 2011.

[61] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *Proc. 9th International Symposium on Spatial and Temporal Databases*, pages 364–381, 2005.

[62] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*, chapter Partitioning Around Medoids, pages 86–89. John Wiley & Sons, 99 edition, 2009.

[63] J. Kennedy. The particle swarm: Social adaptation of knowledge. In *Proc. IEEE International Conference on Evolutionary Computation*, pages 303–308, 1997.

[64] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Proc. Congress on Evolutionary Computation*, volume 3, 1999.

[65] J. Kennedy. Bare bones particle swarms. In *Proc. IEEE Swarm Intelligence Symposium*, pages 80–87, 2003.

[66] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[67] J. Kennedy and R. Mendes. Neighborhood topologies in fully-informed and best-of-neighborhood particle swarms. In *Proc. IEEE International Workshop on Soft Computing in Industrial Applications*, pages 45–50, 2003.

[68] J. F. Kennedy, J. Kennedy, and R. C. Eberhart. *Swarm Intelligence*. Scott Norton, 2011.

[69] J. Kim and P. Bentley. Negative selection and niching by an artificial immune system for network intrusion detection. In *Proc. In Late Breaking Papers at the*

1999 Genetic and Evolutionary Computation Conference, pages 14–158, 1999.

[70] A. V. N. Krishna and M. P. Roy. A spatial regression analysis model for temporal data mining in estimation of traffic data over a busy area. *Georgian Electronic Scientific Journal: Computer Science and Telecommunications*, 3(14):84–88, 2007.

[71] J-B de Lamarck. *Zoological philosophy*, volume 1. Translated by Ian Johnston, 1914.

[72] C. Y. Lee and E. K. Antonsson. Dynamic partitional clustering using evolution strategies. In *Proc. 26th Annual Conference of the IEEE Industrial Electronics Society*, volume 4, pages 2716–2721, 2000.

[73] C. Li and S Yang. A clustering particle swarm optimizer for dynamic optimization. In *Proc. IEEE Congress on Evolutionary Computation*, pages 439–446, 2009.

[74] T. Liu, Y. Zhou, Z. Hu, and Z. Wang. A new clustering algorithm based on artificial immune system. In *Proc. Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, volume 2, pages 347–351, 2008.

[75] Y. Lu, S. Wang, S. Li, and C. Zhou. Particle swarm optimizer for variable weighting in clustering high-dimensional data. *Machine learning*, 82(1):43–70, 2011.

[76] G. Lyengar and A. Lippman. Clustering images using relative entropy for efficient retrieval (1998). In *Proc. International Workshop on Very Low Bitrate Video Coding (VLBV)*, 1998.

[77] N. K. Madavan. Multiobjective optimization using a pareto differential evolution approach. In *Proc. Congress on Evolutionary Computation*, volume 2, pages 1145–1150, 2002.

[78] G. D. Marketos. *Intelligent stock market assistant using temporal data mining*. University of Manchester, 2004.

[79] U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern recognition*, 33(9):1455–1465, 2000.

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

[81] E. Meyer, P. Grussenmeyer, J. P. Perrin, A. Durand, and P. Drap. Intra-site level cultural heritage documentation: Combination of survey, modeling and imagery data in a web information system. In *Proc. 7th International Symposium on Virtual Reality, Archaeology and Cultural Heritage*, 2006.

[82] C. S. Möller-Levet, F. Klawonn, K-H Cho, and O. Wolkenhauer. Fuzzy clustering of short time-series and unevenly distributed sampling points. 2810:330–340, 2003.

[83] O. Nasraoui, F. Gonzalez, C. Cardona, C. Rojas, and D. Dasgupta. A scalable artificial immune system model for dynamic unsupervised learning. *Proc. Genetic and Evolutionary Computation Conference*, pages 219–230, 2003.

[84] F. V. Nepomuceno and A. P. Engelbrecht. A self-adaptive heterogeneous pso for real-parameter optimization. In *Proc. IEEE Congress on Evolutionary Computation*, pages 361–368, June 2013.

[85] M. G. H. Omran, A. P. Engelbrecht, and A. Salman. Dynamic clustering using particle swarm optimization with application in unsupervised image classification. *Transactions on Engineering, Computing and Technology*, 9:199–204, 2005.

[86] M. G. H. Omran, A. P. Engelbrecht, and A. Salman. Bare bones differential evolution. *European Journal of Operational Research*, 196(1):128–139, 2009.

[87] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Springer, 2003.

[88] G. Pampara, A. P. Engelbrecht, and T. Cloete. Cilib: A collaborative framework for computational intelligence algorithms-part i. In *Proc. IEEE International Joint Conference on Neural Networks*, pages 1750–1757, 2008.

[89] A. Panuccio, M. Bicego, and V. Murino. A hidden markov model-based approach

to sequential data clustering. 2396:734–743, 2002.

[90] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, 2002.

[91] K. E. Parsopoulos, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis. Vector evaluated differential evolution for multiobjective optimization. In *Proc. Congress on Evolutionary Computation*, volume 1, pages 204–211, 2004.

[92] J. Patel. Temporal database system. Master's thesis, Department of Computing, Imperial College, University of London,, June 2003.

[93] S. Paterlini and T. Krink. High performance clustering with differential evolution. In *Proc. Congress on Evolutionary Computation*, volume 2, pages 2004–2011, 2004.

[94] S. Paterlini and T. Krink. Differential evolution and particle swarm optimisation in partitional clustering. *Computational Statistics & Data Analysis*, 50(5):1200–1247, 2006.

[95] E. S. Peer, A. P. Engelbrecht, G. Pampara, and B. S. Masiye. Ciclops: computational intelligence collaborative laboratory of pantological software. In *Proc. Swarm Intelligence Symposium*, pages 130–137, 2005.

[96] M. A. Potter and K. A. De Jong. The coevolution of antibodies for concept learning. In *Proc. The Fifth Parallel Problem Solving from Nature*, pages 530–539, 1998.

[97] A. K. Qin and P. N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Proc. The 2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1785–1791, 2005.

[98] R Core Team. Manual: R: A language and environment for statistical computing. Available: http://www.R-project.org/ (Accessed: 5 December 2012), 2014.

[99] S. Ray and R. H. Turi. Determination of number of clusters in k-means clustering application in colour image segmentation. In *Proc. The 4th International*

Conference on Pattern Recognition and Digital Techniques, pages 137–143, 1999.

[100] T. Ng Raymond and H. Jiawei. Efficient and effective clustering methods for spatial data mining. In *Proc. 20th Very Large Databases Conference (VLDB)*, pages 144–155, 1994.

[101] Mark Richards and Dan Ventura. Dynamic sociometry in particle swarm optimization. In *Proc. International Conference on Computational Intelligence and Natural Computing*, pages 1557–1560, 2003.

[102] I. Saha, U. Maulik, and D. Plewczynski. A new multi-objective technique for differential fuzzy clustering. *Applied Soft Computing*, 11(2):2765–2776, 2011.

[103] P. S. Shelokar, V. K. Jayaraman, and B. D. Kulkarni. An ant colony approach for clustering. *Analytica Chimica Acta*, 509(2):187–195, 2004.

[104] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proc. IEEE Congress on Evolutionary Computation*, page 69–73, 1998.

[105] N. Soni and A. Ganatra. Categorization of several clustering algorithms from different perspective: A review. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(8), August 2012.

[106] T. Sousa, A. Silva, and A. Neves. Particle swarm based data mining algorithms for classification tasks. *Parallel Computing*, 30(5):767–783, 2004.

[107] S. Spencer. *The Principles of Biology*, volume 1. Harvard University, 1864.

[108] R. Storn. On the usage of differential evolution for function optimization. In *Proc. Biennial Conference of the North American Fuzzy Information Processing Society*, pages 519–523, 1996.

[109] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. *Science*, 11(4):1–15, 1995.

[110] H. Sun, S. Wang, and Q. Jiang. Fcm-based model selection algorithms for deter-

mining the number of clusters. *Pattern recognition*, 37(10):2027–2037, 2004.

[111] K. Suresh, S. Kundu, and Das S. Abraham A. Ghosh, S. Data clustering using multi-objective differential evolution algorithms. *Fundamenta Informaticae*, 97:381–403, 2009.

[112] Snodgrass R. T., Dasgupt P. D., and Urban J. E. *Temporal Database*. Kluwer Academic Publishers, 1998.

[113] Warren L. T. Clustering of time series data — a survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[114] R. E. Taylor. Dating techniques in archaeology and paleoanthropology. *Analytical Chemistry*, 59(4):317A–331A, 1987.

[115] J. Timmis, M. Neal, and J. Hunt. An artificial immune system for data analysis. *Biosystems*, 55(1–3):143 – 150, 2000.

[116] A. Ushioda. Hierarchical clustering of words. In *Proc. 16th Conference on Computational Linguistics*, volume 2, pages 1159–1162, 1996.

[117] F. van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. In *Proc. IEEE Transactions on Evolutionary Computation*, volume 8, pages 225–239, 2004.

[118] D. W. van der Merwe and A. P. Engelbrecht. Data clustering using particle swarm optimization. In *Proc. The 2003 Congress on Evolutionary Computation*, volume 1, pages 215–220, 2003.

[119] M. Van Der Voort, M. Dougherty, and S. Watson. Combining kohonen maps with arima time series models to forecast traffic flow. *Transportation Research Part C: Emerging Technologies*, 4(5):307–318, 1996.

[120] K. N. Veena and B. P. Vijaya Kumar. Hidden markov model with computational intelligence for dynamic clustering in wireless sensor networks. In *Proc. International Conference on Advances in Computing*, pages 19–30, 2012.

[121] C. J. Veenman, M. J. T. Reinders, and E. Backer. A maximum variance cluster algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1273–1280, 2002.

[122] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series. In *Proc. Workshop on Clustering High Dimensionality Data and Its Applications*, pages 23–30, 2003.

[123] H. Wang, T. Zhang, X. Tang, and Y. Liu. Building a dynamic, large-scale spatio-temporal vector database to support a national spatial data infrastructure in china. *GIScience & Remote Sensing*, 47(1):135–162, 2010.

[124] T. Wang, C. Perng, T. Tao, C. Tang, E. So, C. Zhang, R. Chang, and L. Liu. A temporal data-mining approach for discovering end-to-end transaction flows. In *Proc. IEEE International Conference on Web Services*, pages 37–44, 2008.

[125] D. Wheatley and M. Gillings. *Spatial Technology and Archaeology: The Archaeological Applications of Georgraphical Information Systems*. CRC Press, 2004.

[126] D. Whitley, V. S. Gordon, and K Mathias. Lamarckian evolution, the baldwin effect and function optimization. 866:5–15, 1994.

[127] A. Wismüller, O. Lange, D. R. Dersch, G. L. Leinsinger, K. Hahn, B. Pütz, and D. Auer. Cluster analysis of biomedical image time-series. *International Journal of Computer Vision*, 46(2):103–128, 2002.

[128] S. Wu, A. W-C Liew, H. Yan, and M. Yang. Cluster analysis of gene expression data based on self-splitting and merging competitive learning. *IEEE Transactions on Information Technology in Biomedicine*, 8(1):5–15, 2004.

[129] X. L. Xie and G. Beni. A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):841–847, 1991.

[130] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[131] W. Xu and H. Huang. Research and application of spatio-temporal data mining based on ontology. In *Proc. First International Conference on Innovative Computing, Information and Control*, volume 2, pages 535–538, 2006.

[132] F. Yang, T. Sun, and C. Zhang. An efficient hybrid data clustering method based on k-harmonic means and particle swarm optimization. *Expert Systems with Applications*, 36(6):9847–9852, 2009.

[133] S. Yang, M. Wang, and L. Jiao. A quantum particle swarm optimization. In *Proc. Congress on Evolutionary Computation*, volume 1, pages 320–324, 2004.

[134] A. L. Zadeh, G. J. Klir, and B. Yuan. *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers*. World Scientific Publishing, 1995.

[135] L. A. Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[136] D. Zaharie. A comparative analysis of crossover variants in differential evolution. In *Proc. 2nd International Symposium Advances in Artificial Intelligence and Applications*, pages 171–181.

[137] B. Zhang, H. Meichun, and D. Umeshwar. K-harmonic means-a data clustering algorithm. Technical report, Hewlett-Packard Labs, 1999.

[138] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 515–524, 2002.

[139] A. K. Ziliaskopoulos and S. T. Waller. An internet-based geographic information system that integrates data, models and users for transportation applications. *Transportation Research Part C: Emerging Technologies*, 8(1):427–444, 2000.

Appendix A

Detailed Local Network Neighbourhood Artificial Immune System Parameters

This appendix contains tables describing the parameters of the local network neighbourhood artificial immune system (LNNAIS). In the thesis, the k-independent cooperative data clustering differential evolution (KCDCDynDE) is compared to the LNNAIS algorithm on the same set of problems. Graaff and Engelbrecht [46] optimised the LNNAIS parameters for each of the 225 problems. Tables A.1, A.2 and A.3 provide the parameters used for each pattern migration, centroid migration and cluster migration respectively.

Problem	$\mathcal{B}_{max}$	ρ	ε_{clone}	**Problem**	$\mathcal{B}_{max}$	ρ	ε_{clone}
d=3 f=1 s=1	50	3	5	d=3 f=1 s=2	50	3	5
d=3 f=1 s=3	40	3	5	d=3 f=1 s=4	40	3	5
d=3 f=1 s=5	50	3	5	d=3 f=2 s=1	50	3	5
d=3 f=2 s=2	40	3	5	d=3 f=2 s=3	50	3	5
d=3 f=2 s=4	50	3	5	d=3 f=2 s=5	50	3	5
d=3 f=3 s=1	40	3	5	d=3 f=3 s=2	50	3	5
d=3 f=3 s=3	40	3	5	d=3 f=3 s=4	40	3	5
d=3 f=3 s=5	40	3	5	d=3 f=4 s=1	50	3	5

d=3 f=4 s=2	50	3	5	d=3 f=4 s=3	40	3	5
d=3 f=4 s=4	50	3	5	d=3 f=4 s=5	40	3	5
d=3 f=5 s=1	40	3	5	d=3 f=5 s=2	40	3	5
d=3 f=5 s=3	40	3	5	d=3 f=5 s=4	50	3	5
d=3 f=5 s=5	50	3	5	d=8 f=1 s=1	10	3	5
d=8 f=1 s=2	10	3	5	d=8 f=1 s=3	10	3	5
d=8 f=1 s=4	10	3	5	d=8 f=1 s=5	10	3	5
d=8 f=2 s=1	10	3	5	d=8 f=2 s=2	10	3	5
d=8 f=2 s=3	10	3	5	d=8 f=2 s=4	10	3	5
d=8 f=2 s=5	10	3	5	d=8 f=3 s=1	10	3	5
d=8 f=3 s=2	10	3	5	d=8 f=3 s=3	10	3	5
d=8 f=3 s=4	10	3	5	d=8 f=3 s=5	10	3	5
d=8 f=4 s=1	10	3	5	d=8 f=4 s=2	10	3	5
d=8 f=4 s=3	10	3	5	d=8 f=4 s=4	10	3	5
d=8 f=4 s=5	10	3	5	d=8 f=5 s=1	10	3	5
d=8 f=5 s=2	10	3	5	d=8 f=5 s=3	50	3	5
d=8 f=5 s=4	50	3	5	d=8 f=5 s=5	10	3	5
d=15 f=1 s=1	10	3	5	d=15 f=1 s=2	40	3	5
d=15 f=1 s=3	10	3	5	d=15 f=1 s=4	10	3	5
d=15 f=1 s=5	10	3	5	d=15 f=2 s=1	10	3	5
d=15 f=2 s=2	10	3	5	d=15 f=2 s=3	10	3	5
d=15 f=2 s=4	10	3	5	d=15 f=2 s=5	10	3	5
d=15 f=3 s=1	10	3	5	d=15 f=3 s=2	10	3	5
d=15 f=3 s=3	10	3	5	d=15 f=3 s=4	10	3	5
d=15 f=3 s=5	10	3	5	d=15 f=4 s=1	10	3	5
d=15 f=4 s=2	10	3	5	d=15 f=4 s=3	10	3	5
d=15 f=4 s=4	10	3	5	d=15 f=4 s=5	10	3	5
d=15 f=5 s=1	10	3	5	d=15 f=5 s=2	50	3	5
d=15 f=5 s=3	10	3	5	d=15 f=5 s=4	10	3	5
d=15 f=5 s=5	10	3	5				

Table A.1: Detailed LNNAIS Parameters for Pattern Migration Problems

Problem	$\mathcal{B}_{max}$	ρ	ε_{clone}	**Problem**	$\mathcal{B}_{max}$	ρ	ε_{clone}
d=3 f=1 s=1	40	3	5	d=3 f=1 s=2	10	3	5
d=3 f=1 s=3	10	3	5	d=3 f=1 s=4	40	3	5

d=3 f=1 s=5	40	3	5	d=3 f=2 s=1	50	3	5
d=3 f=2 s=2	40	3	5	d=3 f=2 s=3	10	3	5
d=3 f=2 s=4	30	3	5	d=3 f=2 s=5	40	3	5
d=3 f=3 s=1	40	3	5	d=3 f=3 s=2	50	3	5
d=3 f=3 s=3	40	3	5	d=3 f=3 s=4	40	3	5
d=3 f=3 s=5	40	3	5	d=3 f=4 s=1	40	3	5
d=3 f=4 s=2	50	3	5	d=3 f=4 s=3	40	3	5
d=3 f=4 s=4	50	3	5	d=3 f=4 s=5	40	3	5
d=3 f=5 s=1	40	3	5	d=3 f=5 s=2	30	3	5
d=3 f=5 s=3	40	3	5	d=3 f=5 s=4	40	3	5
d=3 f=5 s=5	50	3	5	d=8 f=1 s=1	30	3	5
d=8 f=1 s=2	10	3	5	d=8 f=1 s=3	50	3	5
d=8 f=1 s=4	30	3	5	d=8 f=1 s=5	30	3	5
d=8 f=2 s=1	40	3	5	d=8 f=2 s=2	40	3	5
d=8 f=2 s=3	40	3	5	d=8 f=2 s=4	40	3	5
d=8 f=2 s=5	30	3	5	d=8 f=3 s=1	10	3	5
d=8 f=3 s=2	40	3	5	d=8 f=3 s=3	10	3	5
d=8 f=3 s=4	40	3	5	d=8 f=3 s=5	10	3	5
d=8 f=4 s=1	40	3	5	d=8 f=4 s=2	30	3	5
d=8 f=4 s=3	10	3	5	d=8 f=4 s=4	10	3	5
d=8 f=4 s=5	10	3	5	d=8 f=5 s=1	10	3	5
d=8 f=5 s=2	50	3	5	d=8 f=5 s=3	50	3	5
d=8 f=5 s=4	10	3	5	d=8 f=5 s=5	10	3	5
d=15 f=1 s=1	10	3	5	d=15 f=1 s=2	10	3	5
d=15 f=1 s=3	30	3	5	d=15 f=1 s=4	40	4	5
d=15 f=1 s=5	10	3	5	d=15 f=2 s=1	10	3	5
d=15 f=2 s=2	10	3	5	d=15 f=2 s=3	10	3	5
d=15 f=2 s=4	50	3	5	d=15 f=2 s=5	10	3	5
d=15 f=3 s=1	10	3	5	d=15 f=3 s=2	10	3	5
d=15 f=3 s=3	50	3	5	d=15 f=3 s=4	10	3	5
d=15 f=3 s=5	10	3	5	d=15 f=4 s=1	40	3	5
d=15 f=4 s=2	30	3	5	d=15 f=4 s=3	30	3	5
d=15 f=4 s=4	10	3	5	d=15 f=4 s=5	10	3	5
d=15 f=5 s=1	10	3	5	d=15 f=5 s=2	10	3	5
d=15 f=5 s=3	10	3	5	d=15 f=5 s=4	40	3	5
d=15 f=5 s=5	10	3	5				

Table A.2: Detailed LNNAIS Parameters for Centroid Migration Problems

Problem	$\mathcal{B}_{max}$	ρ	ε_{clone}	**Problem**	$\mathcal{B}_{max}$	ρ	ε_{clone}
d=3 f=1 s=1	40	3	5	d=3 f=1 s=2	50	3	5
d=3 f=1 s=3	50	3	5	d=3 f=1 s=4	40	3	5
d=3 f=1 s=5	50	3	5	d=3 f=2 s=1	40	3	5
d=3 f=2 s=2	50	3	5	d=3 f=2 s=3	40	3	5
d=3 f=2 s=4	50	3	5	d=3 f=2 s=5	50	3	5
d=3 f=3 s=1	40	3	5	d=3 f=3 s=2	50	3	5
d=3 f=3 s=3	40	3	5	d=3 f=3 s=4	50	3	5
d=3 f=3 s=5	50	3	5	d=3 f=4 s=1	50	3	5
d=3 f=4 s=2	50	3	5	d=3 f=4 s=3	40	3	5
d=3 f=4 s=4	50	3	5	d=3 f=4 s=5	40	3	5
d=3 f=5 s=1	50	3	5	d=3 f=5 s=2	40	3	5
d=3 f=5 s=3	50	3	5	d=3 f=5 s=4	40	3	5
d=3 f=5 s=5	50	3	5	d=8 f=1 s=1	10	3	5
d=8 f=1 s=2	10	3	5	d=8 f=1 s=3	10	3	5
d=8 f=1 s=4	10	3	5	d=8 f=1 s=5	10	3	5
d=8 f=2 s=1	10	3	5	d=8 f=2 s=2	50	3	5
d=8 f=2 s=3	10	3	5	d=8 f=2 s=4	10	3	5
d=8 f=2 s=5	10	3	5	d=8 f=3 s=1	10	3	5
d=8 f=3 s=2	10	3	5	d=8 f=3 s=3	10	3	5
d=8 f=3 s=4	50	3	5	d=8 f=3 s=5	40	3	5
d=8 f=4 s=1	10	3	5	d=8 f=4 s=2	10	3	5
d=8 f=4 s=3	10	3	5	d=8 f=4 s=4	10	3	5
d=8 f=4 s=5	10	3	5	d=8 f=5 s=1	10	3	5
d=8 f=5 s=2	40	3	5	d=8 f=5 s=3	10	3	5
d=8 f=5 s=4	10	3	5	d=8 f=5 s=5	10	3	5
d=15 f=1 s=1	10	3	5	d=15 f=1 s=2	40	3	5
d=15 f=1 s=3	10	3	5	d=15 f=1 s=4	10	3	5
d=15 f=1 s=5	10	3	5	d=15 f=2 s=1	10	3	5
d=15 f=2 s=2	10	3	5	d=15 f=2 s=3	40	3	5
d=15 f=2 s=4	10	3	5	d=15 f=2 s=5	10	3	5
d=15 f=3 s=1	40	3	5	d=15 f=3 s=2	10	3	5
d=15 f=3 s=3	10	3	5	d=15 f=3 s=4	10	3	5

d=15 f=3 s=5	10	3	5	d=15 f=4 s=1	10	3	5
d=15 f=4 s=2	10	3	5	d=15 f=4 s=3	10	3	5
d=15 f=4 s=4	10	3	5	d=15 f=4 s=5	10	3	5
d=15 f=5 s=1	10	3	5	d=15 f=5 s=2	10	3	5
d=15 f=5 s=3	10	3	5	d=15 f=5 s=4	10	3	5
d=15 f=5 s=5	10	3	5				

Table A.3: Detailed LNNAIS Parameters for Cluster Migration Problems

Appendix B

Detailed K-independent Cooperative Data Clustering Dynamic Differential Evolution Parameters

This appendix contains tables describing the parameters of the k-independent cooperative data clustering differential evolution (KCDCDynDE). In the thesis the KCDCDynDE is compared to the local network neighbourhood artificial immune system (LNNAIS) algorithm. Graaff and Engelbrecht [46] empirically optimised the LNNAIS parameters for each of the 225 problems. The parameters of the KCDCDynDE were empirically optimised for each of the 225 problems. The scale parameter and crossover probability were both set to 0.5, and 10% of the population was set to Brownian individuals. Tables B.1, B.2 and B.3 provide the exclusion radius and crossover probability values used for each pattern migration, centroid migration and cluster dimension respectively.

Problem	**Exclusion radius**	**Convergence radius**	**Problem**	**Exclusion radius**	**Convergence radius**
d=3 f=1 s=1	3.5	2.0	d=3 f=1 s=2	3.5	2.0
d=3 f=1 s=3	3.0	3.0	d=3 f=1 s=4	3.0	2.5
d=3 f=1 s=5	3.5	2.25	d=3 f=2 s=1	3.5	2.25
d=3 f=2 s=2	3.5	2.0	d=3 f=2 s=3	3.5	2.0

d=3 f=2 s=4	3.5	2.0	d=3 f=2 s=5	3.5	2.0
d=3 f=3 s=1	3.5	2.0	d=3 f=3 s=2	3.5	2.0
d=3 f=3 s=3	3.5	2.0	d=3 f=3 s=4	3.5	2.0
d=3 f=3 s=5	3.5	2.0	d=3 f=4 s=1	3.5	2.0
d=3 f=4 s=2	3.5	2.0	d=3 f=4 s=3	3.5	2.0
d=3 f=4 s=4	3.5	2.0	d=3 f=4 s=5	3.5	2.0
d=3 f=5 s=1	3.5	2.0	d=3 f=5 s=2	3.5	2.0
d=3 f=5 s=3	3.5	2.0	d=3 f=5 s=4	3.5	2.0
d=3 f=5 s=5	3.5	2.0	d=8 f=1 s=1	4.75	1.25
d=8 f=1 s=2	4.75	1.25	d=8 f=1 s=3	4.75	1.25
d=8 f=1 s=4	4.75	1.25	d=8 f=1 s=5	4.75	1.25
d=8 f=2 s=1	5.0	1.25	d=8 f=2 s=2	5.0	1.25
d=8 f=2 s=3	4.75	1.25	d=8 f=2 s=4	4.75	1.25
d=8 f=2 s=5	5.0	1.5	d=8 f=3 s=1	4.75	1.25
d=8 f=3 s=2	4.75	1.25	d=8 f=3 s=3	4.75	1.25
d=8 f=3 s=4	4.75	1.25	d=8 f=3 s=5	5.0	1.25
d=8 f=4 s=1	5.0	1.25	d=8 f=4 s=2	4.75	1.25
d=8 f=4 s=3	5.0	1.25	d=8 f=4 s=4	5.0	1.25
d=8 f=4 s=5	4.75	1.25	d=8 f=5 s=1	4.75	1.25
d=8 f=5 s=2	5.0	1.25	d=8 f=5 s=3	4.75	1.25
d=8 f=5 s=4	4.75	1.25	d=8 f=5 s=5	4.75	1.25
d=15 f=1 s=1	5.5	2.0	d=15 f=1 s=2	5.25	2.25
d=15 f=1 s=3	5.0	1.75	d=15 f=1 s=4	5.25	2.25
d=15 f=1 s=5	5.0	1.75	d=15 f=2 s=1	5.5	2.15
d=15 f=2 s=2	5.0	1.75	d=15 f=2 s=3	5.0	2.0
d=15 f=2 s=4	5.5	2.25	d=15 f=2 s=5	5.0	2.0
d=15 f=3 s=1	5.25	2.0	d=15 f=3 s=2	5.25	2.25
d=15 f=3 s=3	5.0	1.75	d=15 f=3 s=4	5.25	2.0
d=15 f=3 s=5	4.0	1.5	d=15 f=4 s=1	5.0	1.75
d=15 f=4 s=2	5.5	1.8	d=15 f=4 s=3	5.25	2.0
d=15 f=4 s=4	5.5	2.0	d=15 f=4 s=5	5.5	2.0
d=15 f=5 s=1	5.5	2.25	d=15 f=5 s=2	5.25	2.25
d=15 f=5 s=3	5.5	2.25	d=15 f=5 s=4	5.5	2.0
d=15 f=5 s=5	5.5	2.25			

Table B.1: Detailed KCDCDynDE Parameters for Pattern Migration Problems

Problem	**Exclusion radius**	**Convergence radius**	**Problem**	**Exclusion radius**	**Convergence radius**
d=3 f=1 s=1	3.5	2.0	d=3 f=1 s=2	3.5	0.75
d=3 f=1 s=3	3.5	3.5	d=3 f=1 s=4	3.0	1.25
d=3 f=1 s=5	3.5	2.0	d=3 f=2 s=1	3.5	2.0
d=3 f=2 s=2	3.5	2.0	d=3 f=2 s=3	3.5	2.0
d=3 f=2 s=4	3.5	2.0	d=3 f=2 s=5	3.5	0.75
d=3 f=3 s=1	3.5	2.0	d=3 f=3 s=2	3.0	3.0
d=3 f=3 s=3	3.5	2.0	d=3 f=3 s=4	3.5	2.0
d=3 f=3 s=5	3.5	2.0	d=3 f=4 s=1	3.5	2.0
d=3 f=4 s=2	3.0	2.0	d=3 f=4 s=3	3.5	2.0
d=3 f=4 s=4	3.25	2.25	d=3 f=4 s=5	3.5	2.0
d=3 f=5 s=1	3.5	2.0	d=3 f=5 s=2	3.5	2.0
d=3 f=5 s=3	3.5	2.0	d=3 f=5 s=4	3.5	2.0
d=3 f=5 s=5	3.5	2.0	d=8 f=1 s=1	4.75	1.25
d=8 f=1 s=2	5.0	1.0	d=8 f=1 s=3	4.0	1.5
d=8 f=1 s=4	4.75	1.25	d=8 f=1 s=5	5.0	1.5
d=8 f=2 s=1	5.5	2.0	d=8 f=2 s=2	4.75	1.25
d=8 f=2 s=3	4.0	1.5	d=8 f=2 s=4	5.0	1.25
d=8 f=2 s=5	4.75	1.25	d=8 f=3 s=1	4.75	1.25
d=8 f=3 s=2	4.0	1.5	d=8 f=3 s=3	4.0	1.5
d=8 f=3 s=4	5.5	2.0	d=8 f=3 s=5	5.5	2.0
d=8 f=4 s=1	4.75	1.25	d=8 f=4 s=2	4.75	1.25
d=8 f=4 s=3	4.75	1.25	d=8 f=4 s=4	5.5	2.0
d=8 f=4 s=5	4.75	1.25	d=8 f=5 s=1	4.75	1.25
d=8 f=5 s=2	4.75	1.25	d=8 f=5 s=3	4.75	1.25
d=8 f=5 s=4	4.75	1.25	d=8 f=5 s=5	5.0	1.25
d=15 f=1 s=1	4.75	2.5	d=15 f=1 s=2	5.0	1.75
d=15 f=1 s=3	4.75	2.5	d=15 f=1 s=4	5.25	2.25
d=15 f=1 s=5	5.5	2.0	d=15 f=2 s=1	4.75	1.5
d=15 f=2 s=2	5.5	2.25	d=15 f=2 s=3	4.75	2.5
d=15 f=2 s=4	5.25	2.0	d=15 f=2 s=5	4.75	2.5
d=15 f=3 s=1	5.25	2.0	d=15 f=3 s=2	5.5	2.0
d=15 f=3 s=3	4.75	2.0	d=15 f=3 s=4	5.5	2.0
d=15 f=3 s=5	4.75	2.0	d=15 f=4 s=1	5.0	1.5
d=15 f=4 s=2	4.7	1.7	d=15 f=4 s=3	4.75	2.5

d=15 f=4 s=4	4.75	2.0	d=15 f=4 s=5	5.5	2.15
d=15 f=5 s=1	5.5	2.25	d=15 f=5 s=2	4.7	1.7
d=15 f=5 s=3	5.5	2.15	d=15 f=5 s=4	5.5	2.0
d=15 f=5 s=5	5.5	2.0			

Table B.2: Detailed KCDCDynDE Parameters for Centroid Migration Problems

Problem	**Exclusion radius**	**Convergence radius**	**Problem**	**Exclusion radius**	**Convergence radius**
d=3 f=1 s=1	3.5	3.5	d=3 f=1 s=2	3.0	3.0
d=3 f=1 s=3	3.0	2.0	d=3 f=1 s=4	3.5	2.0
d=3 f=1 s=5	3.5	2.0	d=3 f=2 s=1	3.5	2.0
d=3 f=2 s=2	3.5	2.0	d=3 f=2 s=3	3.0	3.0
d=3 f=2 s=4	3.0	2.5	d=3 f=2 s=5	3.25	2.25
d=3 f=3 s=1	3.5	2.0	d=3 f=3 s=2	3.5	2.5
d=3 f=3 s=3	3.5	2.0	d=3 f=3 s=4	3.25	2.5
d=3 f=3 s=5	5.5	2.0	d=3 f=4 s=1	3.25	3.25
d=3 f=4 s=2	3.5	3.5	d=3 f=4 s=3	3.5	2.0
d=3 f=4 s=4	3.5	2.0	d=3 f=4 s=5	3.5	2.5
d=3 f=5 s=1	3.5	2.0	d=3 f=5 s=2	3.0	2.5
d=3 f=5 s=3	3.5	2.0	d=3 f=5 s=4	3.5	2.0
d=3 f=5 s=5	3.5	3.25	d=8 f=1 s=1	4.75	1.25
d=8 f=1 s=2	4.75	1.25	d=8 f=1 s=3	4.75	1.25
d=8 f=1 s=4	5.0	1.25	d=8 f=1 s=5	5.0	0.9
d=8 f=2 s=1	4.75	1.25	d=8 f=2 s=2	5.0	1.5
d=8 f=2 s=3	4.85	1.25	d=8 f=2 s=4	4.75	1.25
d=8 f=2 s=5	4.75	1.25	d=8 f=3 s=1	5.0	1.25
d=8 f=3 s=2	4.75	1.25	d=8 f=3 s=3	4.75	1.25
d=8 f=3 s=4	4.75	1.25	d=8 f=3 s=5	4.0	1.0
d=8 f=4 s=1	4.75	1.25	d=8 f=4 s=2	4.75	1.25
d=8 f=4 s=3	4.0	1.0	d=8 f=4 s=4	4.75	1.25
d=8 f=4 s=5	4.75	1.25	d=8 f=5 s=1	4.75	1.25
d=8 f=5 s=2	4.75	1.25	d=8 f=5 s=3	4.75	1.25
d=8 f=5 s=4	4.75	1.25	d=8 f=5 s=5	4.75	1.25
d=15 f=1 s=1	5.25	2.0	d=15 f=1 s=2	5.5	2.25
d=15 f=1 s=3	4.7	1.7	d=15 f=1 s=4	4.5	1.75

d=15 f=1 s=5	4.7	1.7	d=15 f=2 s=1	5.25	2.0
d=15 f=2 s=2	4.9	1.8	d=15 f=2 s=3	5.5	2.15
d=15 f=2 s=4	5.0	1.65	d=15 f=2 s=5	5.0	1.5
d=15 f=3 s=1	5.5	2.0	d=15 f=3 s=2	4.9	1.8
d=15 f=3 s=3	5.5	2.0	d=15 f=3 s=4	5.0	1.5
d=15 f=3 s=5	4.75	1.75	d=15 f=4 s=1	4.9	1.8
d=15 f=4 s=2	4.7	1.7	d=15 f=4 s=3	4.75	1.75
d=15 f=4 s=4	4.75	1.75	d=15 f=4 s=5	3.5	1.25
d=15 f=5 s=1	4.5	1.75	d=15 f=5 s=2	5.0	1.85
d=15 f=5 s=3	5.0	1.5	d=15 f=5 s=4	4.5	1.5
d=15 f=5 s=5	5.0	1.5			

Table B.3: Detailed KCDCDynDE Parameters for Cluster Migration Problems

Appendix C

Example XML for K-independent Cooperative Data Clustering Dynamic Differential Evolution for CIlib

This appendix contains an example XML file for running in CIlib.

```
<algorithms>
    <algorithm id="multiPopulationDE" class="algorithm.population.CooperativeMultipopulationAlgorithm">
        <multiSwarmIterationStrategy
        class="clustering.de.iterationstrategies.multipopulation.KIndependentCooperativeDynDEIterationStrategy">
            <convergenceRadius class="controlparameter.ConstantControlParameter" parameter="4.5"/>
            <exclusionRadius class="controlparameter.ConstantControlParameter" parameter="2.5"/>
            <percentageReplaceableIndividuals class="controlparameter.ConstantControlParameter" parameter="10"/>
        </multiSwarmIterationStrategy>
        <addStoppingCondition class="stoppingcondition.MeasuredStoppingCondition" target="1000"/>
        <addPopulationBasedAlgorithm id="clusteringDE" class="clustering.DataClusteringEC">
            <addStoppingCondition class="stoppingcondition.MeasuredStoppingCondition" target="1"/>
            <iterationStrategy class="clustering.de.iterationstrategies.StandardClusteringDEIterationStrategy">
                <boundaryConstraint class="problem.boundaryconstraint.CentroidBoundaryConstraint">
                    <delegate class="problem.boundaryconstraint.ClampingBoundaryConstraint"/>
                </boundaryConstraint>
            </iterationStrategy>
            <initialisationStrategy class="algorithm.initialisation.DataDependantPopulationInitializationStrategy">
                <delegate class="algorithm.initialisation.ClonedPopulationInitialisationStrategy"/>
                <entityNumber value="50"/>
                <entityType class="clustering.entity.ClusterIndividual">
                    <initialisationStrategy class="entity.initialization.RandomBoundedInitializationStrategy"/>
                    <centroidInitialisationStrategy class="entity.initialization.StandardCentroidInitializationStrategy"/>
                    <updateStrategy class="ec.update.clustering.StandardClusteringDEUpdateStrategy">
                        <trialVectorCreationStrategy class="entity.operators.creation.RandCreationStrategy">
                            <scaleParameter class="controlparameter.ConstantControlParameter" parameter="0.5"/>
                        </trialVectorCreationStrategy>
                        <crossoverStrategy class="entity.operators.crossover.de.DifferentialEvolutionBinomialCrossover">
                            <crossoverPointProbability class="controlparameter.ConstantControlParameter" parameter="0.5"/>
                        </crossoverStrategy>
                    </updateStrategy>
                </entityType>
            </initialisationStrategy>
        </addPopulationBasedAlgorithm>
        <addPopulationBasedAlgorithm id="clusteringDE_2" class="clustering.DataClusteringEC">
            ...
        </addPopulationBasedAlgorithm>
        ...
    </algorithm>
</algorithms>
```

Figure C.1: Example of Algorithm Section of XML for Running the

```
<problems>
    <problem id="quantization_d3_s1_f1" class="problem.QuantizationErrorMinimizationProblem" domain="R(-10:10)"
     sourceURL="<location of dataset>/coinDataset.arff">
        <window class="clustering.SlidingWindow" slideFrequency="1000" >
            <extraWindowSize value="4"/>
            <extraWindowSize value="259"/>
            <extraWindowSize value="570"/>
            <extraWindowSize value="29"/>
            <extraWindowSize value="4"/>
        </window>
    </problem>
</problems>

<measurements id="fitness" class="simulator.MeasurementSuite" resolution="10">
    <addMeasurement class="measurement.multiple.TotalNumberOfClusters"/>
    <addMeasurement class="measurement.clustervalidity.RayTuriValidityIndex"/>
    <addMeasurement class="measurement.clustervalidity.InterClusterDistance"/>
    <addMeasurement class="measurement.clustervalidity.IntraClusterDistance"/>
    <addMeasurement class="measurement.multiple.ClusteringClassificationOutput"/>
</measurements>

<simulations>
    <simulation samples="1">
        <algorithm idref="multiPopulationDE"/>
        <problem idref="quantization_d3_s1_f1"/>
        <measurements idref="fitness" />
        <output format="TXT" file="data/roman_coin_clusters_1.txt" />
    </simulation>
    ...
</simulations>
```

Figure C.2: Example of Problem Definition Section of XML for Running the

Appendix D

Acronyms

This appendix contains a list of acronyms used throughout the thesis. The purpose of the appendix is to have an easy reference to all acronyms. The acronyms are alphabetically ordered, where the acronym's meaning is placed alongside the acronym, which is in bold.

ACDE	automatic clustering differential evolution.
AI	Artificial Intelligence.
AIS	artificial immune system.
ALC	artificial lymphocytes.
ANN	Artificial Neural Network.
ARIMA	autoregressive integrated moving average.
CDCDynDE	cooperative dynamic data clustering differential evolution.
CDCPSO	cooperative data clustering particle swarm optimisation.
CI	computational intelligence.
CILib	Computational Intelligence Library.
CMDCPSO	cooperative multi-swarm data clustering particle swarm optimisation.

CPSO	cooperative particle swarm optimisation.
DCDE	data clustering differential evolution.
DCDynDE	dynamic data clustering differential evolution.
DCPSO	data clustering particle swarm optimisation.
DE	differential evolution.
DynDE	dynamic differential evolution.
EC	evolutionary computation.
CMDCPSO	elitist cooperative multi-swarm data clustering particle swarm optimisation.
FS	fuzzy systems.
FSTS	fuzzy time-series.
GA	genetic algorithm.
GIS	geographical information system.
HMM	hidden Markov model.
KCDCDynDE	K-independent cooperative data clustering differential evolution.
KSOM-NN	Kohonen self organisation map neural network.
LNNAIS	local network neighbourhood artificial immune system.
MDCPSO	multi-swarm data clustering particle swarm optimisation.
MPSO	multi-swarm particle swarm optimisation.

MSE	mean squared error.
PSO	particle swarm optimisation.
DCDE	re-initialising data clustering differential evolution.
RDCPSO	reinitialising data clustering particle swarm optimisation.
RDE	re-initialising differential evolution.
SI	swarm intelligence.
SOM	self-organising map.

Appendix E

Symbols

This appendix summarises all the symbols used throughout the thesis. The symbols are organised by the chapter in which they first appear and are presented in two columns, the first containing the symbol and the second containing the description of the symbol.

E.1 Chapter 2: Literature Study

K	Total number of cluster centroids

E.2 Chapter 3: Performance Measures

J_{iter}	Inter-cluster distance
k	The index of a cluster
N	The total dimensions of a vector
$d(\mathbf{c}_k, \mathbf{c}_{k_2})$	The Euclidean distance between cluster centroid $\mathbf{c}_k$ and cluster centroid $\mathbf{c}_{k_2}$
$\mathbf{c}_k$	The cluster centroid of the kth cluster

J_{intra}	Intra-cluster distance
C_k	The kth cluster
$\mathbf{z}_p$	The pth data pattern
$\|P\|$	The total number of data patterns in the dataset
Q_D	Validity index
$\upsilon(C_k)$	Cluster quality for Davis Bouldin validity index
$\|C_k\|$	The total number of data patterns in cluster C_k.
u_{kp}^m	The fuzzy membership of a pattern to a cluster
$b(\mathbf{z}_p)$	The average dissimilarity between a data pattern and all data patterns in its closest cluster
$a(\mathbf{z}_p)$	The average dissimilarity between a data pattern and all other data patterns in its cluster
J_e	Quantization error
SE	Squared error
$hMSE$	Heuristic mean squared error
$XB_i(c)$	Xie-Benni validity index
ϵ	Very small constant greater than zero used to prevent division by zero
∞	Infinity

E.3 Chapter 4: Artificial Datasets

t	Time-step when a change occurs
T	The total number of time-steps or iterations
f	The frequency of change

a	The amplitude used to generate the artificial clusters
N	The dimensionality of the data patterns to be generated
$c_{k,n}$	Dimension n of centroid c_k
$\sigma_{k,n}$	The compactness of a cluster in dimension n
$x_{k,n}$	The offset from centroid $c_{k,n}$ in dimension n
φ	The radius of the (N-1)-dimensional sphere used to project the cluster centroid
$\mathbf{m}$	The middle point of the (N-1)-dimensional sphere used to project the cluster centroid
θ_k	Angle vector representing the centroid to be projected on the (N-1)-dimensional sphere
t_s	The resulting scaled value
t_u	The value to be scaled
$t_{u,min}$	The minimum value of the unscaled dataset
$t_{u,max}$	The maximum value of the unscaled dataset
$t_{s,max}$	The new minimum value t_u is being scaled to
$t_{s,min}$	The new maximum value t_u is being scaled to

E.4 Chapter 5: Particle Swarm Optimization Algorithm for Temporal Data Clustering

n	The dimensionality of the problem
v_{ij}	Dimension j of the velocity of particle i
c_1	Cognitive acceleration coefficient
c_2	Social acceleration coefficient

y_{ij}	Dimension j of the personal best value of particle i
$\mathbf{x}_i$	A particle's position
x_{ij}	Dimension j for the position of particle i
$\hat{y}_j$	Dimension j of the global best particle
r	Random value sampled from a uniform distribution $U(0, 1)$
w	The inertia weight
x_i	A particle containing centroids
$\mathbf{x}_{ij}$	Dimension j for the position of the centroid holding particle i
$\hat{\mathbf{y}}_j$	Dimension j of the global best centroid holding particle
N	Size of the population
P	Collection of data patterns
S_k	The kth swarm or population
s	Total swarms or populations
S	Swarm or population
r_{excl}	Exclusion radius
r_{conv}	Convergence radius
$b(k, S_k.\hat{\mathbf{y}}_i)$	Context particle

E.5 Chapter 6: Differential Evolution Algorithm for Temporal Data Clustering

$\mathbf{u}_i$	Trial vector for the ith individual
$\mathbf{x}_{i_1}$	Target vector for the ith individual

$\mathbf{x}_{i_2}$	Randomly selected individual from the population different from the target vector
$\mathbf{x}_{i_3}$	Randomly selected individual from the population different from the target vector and $\mathbf{x}_{i_2}$
β	Scaling factor
$\mathbf{x}'_i$	Offspring i
$\mathbf{x}'_{ij}$	Dimension j of offspring i
$\mathbf{x}_{ij}$	Dimension j of the parent individual i
J	Collection of crossover points
$\mathbf{w}(x_i j)$	Brownian individual adaptation
$N(0, \sigma)$	Gaussian noise
σ	Standard deviation

E.6 Chapter 7: Local Network Neighbourhood Artificial Immune System Overview

$a(\mathbf{z}_p, \mathbf{x}_i)$	Affinity of pattern $\mathbf{z}_p$ to ALC $\mathbf{x}_i$
$a(i, p)$	The normalised affinity
m	Minimum distance between an antigen pattern and an ALC
$\mathbf{x}_i$	The ith ALC
$\mathbf{x}_j$	ALC in the neighbourhood of $\mathbf{x}_i$
N_i	Neighbourhood size
$na(\mathbf{x}_i)$	The network affinity for ALC $\mathbf{x}_i$
$na(i, p)$	The normalised network affinity

NAT	Network affinity threshold
A	User defined constant used to control connectivity, where $0 < A < 1$
$ns(\mathbf{x}_i)$	The network suppression for ALC $\mathbf{x}_i$
$sl(\mathbf{z}_p, \mathbf{x}_i)$	The stimulation level for ALC $\mathbf{x}_i$ when exposed to antigen $\mathbf{z}_p$
e_x	The clonal rate
G	A user defined constant for the clonal rate
$THRESHOLD$	User predefined threshold for cloning
$\mathbf{x}_i$	The ith ALC
$\mathbf{w}_i$	The ALC with the highest calculated affinity
Cl_h	Collection of antigen mutated clones
$\mathbf{a}'_g$	An antigen mutated clone
$\|Cl_h\|$	The size of the collection of antigen mutated clones
$a * (\mathbf{w}_h, \mathbf{a}'_g, Cl_h)$	The normalised affinity of antigen mutated clones
a_{max}	The maximum affinity between the ALC and its antigen mutated clones
$\mathbf{w}'_h$	Mutated ALC clone

Appendix F

Derived Publications

This appendix contains the publications derived from the research done for this thesis. The publications attached include:

- Georgieva, K. and Engelbrecht, A. P., "A cooperative multi-population approach to clustering temporal data", *IEEE Congress on Evolutionary Computation*, pp.1983-1991, June 2013 (Published)
- Georgieva, K. and Engelbrecht, A. P., "Dynamic Differential Evolution Algorithm for Clustering Temporal Data", *9th International Conference on Large-Scale Scientific Computations*, June 2013 (Published)
- Georgieva, K. and Engelbrecht, A. P., "Cooperative DynDE for Temporal Data Clustering", *IEEE World Congress on Computational Intelligence*, July 2014 (Accepted)

A Cooperative Multi-population Approach to Clustering Temporal Data

Kristina Georgieva
CIRG, Department of
Computer Science
University of Pretoria
Email: kristina.s.georgieva@gmail.com

Andries P. Engelbrecht
CIRG, Department of
Computer Science
University of Pretoria
Email: engel@cs.up.ac.za

***Abstract*—In temporal environments, population-based data clustering algorithms suffer when changes in the data occur during the clustering process. Diversity of the population is lost and memory of the individuals of the population is outdated, making the clusters found before the change non-optimal. This paper proposes a new particle swarm optimisation alternative to clustering temporal data. It combines the dynamic properties of the multi-swarm particle swarm optimisation algorithm with the multi-objective properties of the cooperative particle swarm optimisation algorithm. The proposed alternative is compared to various existing data clustering algorithms which are shortly described in the paper and the results are discussed, including a comparison of four performance measures relevant to the clustering of data.**

I. Introduction

Clustering [1] refers to the grouping of similar objects together such that relationships among these objects can be determined and evaluated. Data clustering has been performed using k-means [2], fuzzy c-means [3], differential evolution [4], artificial immune systems [5], neural networks [6] and particle swarm optimisation [7] [8]. All these algorithms have been used to cluster static data. This paper focuses on evaluating particle swarm optimisation algorithms applied to the problem of clustering temporal data.

Particle swarm optimisation (PSO) algorithms have previously been developed and used for the clustering of static-data [7][8]. A temporal dataset displays a different behaviour to a static one. The dataset changes as time passes, making it more complex to cluster. The complexity arises from the changes that occur in the data, which make the population's current and previous positions no longer optimal. As time passes, the population becomes less diverse making it more difficult for the particles to explore the search space for the new optimal solutions.

In this research, four algorithms have been evaluated on temporal datasets, namely the standard data clustering PSO, the reinitialising data clustering PSO, the multi-swarm PSO and the cooperative PSO. Problems such as the lack of re-diversification of the population and the lack of information exchange between populations in the multi-swarm based algorithms were encountered when evaluating these four algorithms. This paper aims at remedying these problems by combining two of the algorithms into a new clustering algorithm called the cooperative-multipopulation data clustering PSO.

II. Particle Swarm optimisation

The particle swarm optimisation (PSO) algorithm [9] is a population based algorithm modelling the social behaviour of birds. It optimises a solution by adapting the position of "particles" across the search space. These particle positions are changed by giving each dimension of the particle a velocity value, a value determined by a social component and a cognitive component. The social component refers to the influence that the best position found within the entire population or a portion of it has on the new velocity values, while the cognitive component refers to the influence that the best position found by the particle being adapted has over its own velocity. Algorithm 1 adapts the velocity of each particle using [10]

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)] \quad (1)$$

,
and changes their position using [9]

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (2)$$

,
where w is the inertia weight (how much the previous velocity of a particle will contribute to the new velocity), $v_{ij}(t+1)$ is the velocity value for particle i at dimension j at time step $t+1$, c_1 is the influence of the cognitive component, c_2 is the influence of the social component, y_{ij} is the personal best value of particle i for dimension j, x_{ij} is the current position of particle i for dimension j, $\hat{y}_j$ is the global best particle's position for dimension j and r_{1j} and r_{2j} are two random values for each dimension from a sampled from two independent uniform distributions U(0,1), $x_{ij}(t+1)$ is particle i's new position for dimension j at time step $t+1$.

III. Standard Data Clustering PSO

This paper chooses to evaluate Van der Merwe and Engelbrecht's data clustering PSO [7] as the principal PSO algorithm for clustering data, and refers to this PSO as the standard data clustering PSO. The standard clustering PSO adds a

Algorithm 1 PSO

Initialize a swarm of N n-dimensional particles as P
while stopping condition has not been reached **do**
  for each particle $\mathbf{x}_i$ in P **do**
    if fitness($\mathbf{x}_i$) $<$ fitness($\mathbf{y}_i$) **then**
      $\mathbf{y}_i = \mathbf{x}_i$
    end if
    if fitness($\mathbf{y}_i$) $<$ fitness($\hat{\mathbf{y}}$) **then**
      $\hat{\mathbf{y}} = \mathbf{y}_i$
    end if
  end for
  for each particle $\mathbf{x}_i$ in P **do**
    Update velocity using equation (1)
    Update position using equation (2)
  end for
end while

few changes to the particle swarm optimisation algorithm shown in Algorithm 1 in order for the algorithm to take into consideration a dataset when determining the fitness value of a particle. The first change is the representation of a particle. While the particle swarm optimisation algorithm described by Kennedy and Eberhart [9] represents a particle as a vector, a particle of the standard data clustering PSO holds a set of vectors, as shown in Figure 1. Each vector represents a cluster centroid. The aim of this PSO is to adapt the positions of these centroids to the center of each cluster, such that the quantization error is minimised.

The second change involves the fact that the dataset needs to influence the results in some way. The fitness function ensures this by first assigning each data pattern to the centroid closest to it. Then, by calculating the quantization error,

$$J_e = \frac{\sum_{k=1}^{K} \frac{\sum_{\forall z \in C_k} d(z, c_k)}{|C_k|}}{K} \quad (3)$$

each particle is assigned a fitness value depending on how close its cluster centroids are to the real clusters, where $\mathbf{z}$ is a data pattern, C_k is the set of data patterns assigned to a cluster, $d(\mathbf{z}, \mathbf{c_k})$ is the Euclidean distance between data pattern $\mathbf{z}$ and cluster centroid $\mathbf{c_k}$, $|C_k|$ is the total number of patterns assigned to the centroid and K is the total number of centroids. The quantization error used in the standard data clustering PSO allows for division by zero. This is due to dividing by the total number of data patterns assigned to a centroid, as centroids that are far from all patterns may result in no patterns being assigned to them, making this number zero. In order to use the algorithm without changing the behaviour, if a division by zero was encountered, the fitness of the particle was approximated to infinity.

The standard data clustering PSO then uses the dataset to determine the fitness of each particle and performs the same velocity update and position update functions that the Standard PSO does. The standard data clustering PSO is shown in Algorithm 2 below.

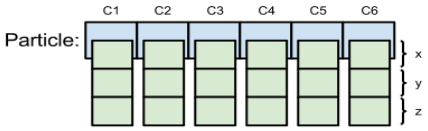


Fig. 1: Particle Representation a 6 centroid clustering problem where x,y and z are the attributes of the data

Algorithm 2 Standard Data Clustering PSO

Initialize a swarm, P, of N Kxn-dimensional particles
while stopping condition is not reached **do**
  for each particle x_i in the population **do**
    for each data pattern $\mathbf{z}_p$ **do**
      calculate the Euclidean distance between $\mathbf{z}_p$ and all centroids
      assign $\mathbf{z}_p$ to the centroid that resulted in the smallest distance
    end for
    Calculate the particle's fitness using equation (3)
    Update the global and personal best positions
    Update velocity using equation (1)
    Update position using equation (2)
  end for
end while

IV. Particle Swarm Optimisation Algorithms for Clustering Temporal Data

The standard data clustering PSO described was developed to cluster stationary data. As this paper focuses on temporal data rather than stationary data, a few algorithms designed for dynamic environments were evaluated and compared against each other in order to determine the most appropriate one for the clustering of temporal data. These algorithms are shortly described below and include the reinitialising data clustering PSO, the multi-swarm PSO, and the cooperative PSO.

A. Reinitialising Data Clustering PSO

The reinitialising PSO [11] is a standard data clustering PSO where a portion of, or the whole population is re-initialised as soon as a change occurs in the environment. This means that a portion of the population is placed in random positions around the search space in order to increase diversity.

B. Multi-swarm Data Clustering PSO

The original multi-swarm PSO described by Blackwell [12] uses a set of populations, each optimising one of the multiple solutions. In the case of data clustering, the particles need information from all cluster centroids in order to calculate their fitness value. For this reason, the original multi-swarm algorithm, where each population optimises one solution, could not be implemented. An alternative was evaluated. Here, Blackwell's multi-swarm was implemented as shown in Algorithm 3, but instead of each population optimising one centroid, each population attempts to optimise all centroids,

using the same representation of a particle as the standard data clustering PSO. This results in a set of populations all working in parallel on the same problem around different areas of the search space. In the end, the best solution out of each population's best solution is the final solution.

Algorithm 3 Data Clustering Multi-swarm PSO

```
Initialize s swarms S_k, k = 1, ..., s
Initialise r, the exclusion radius
Initialise r_conv, the convergence radius
while stopping condition has not been reached do
    if all swarms have a diameter which is smaller than r_conv then
        Reinitialize the worst one by placing its individuals in random positions of the search space
    end if
    for each swarm S_k do
        Perform an iteration of the standard data clustering PSO for S_k
    end for
    for each swarm S_k do
        for each other swarm S_l do
            if distance between ŷ_k and ŷ_l < r then
                if fitness(ŷ_k) < fitness(ŷ_l) then
                    Reinitialize S_k by placing its individuals in random positions of the search space
                else
                    Reinitialize S_j by placing its individuals in random positions of the search space
                end if
            end if
        end for
    end for
end while
```

The algorithm acts in the same way as the one described by Blackwell, by using repulsion and anti-convergence methods to keep the swarms diverse. If two swarms' global best values are within an exclusion radius from each other, one of these populations is re-initialised. In the same manner, if all the swarms are converging, which is determined by the diameter of a swarm being within a convergence radius, one of the populations is re-initialised.

C. Cooperative Data Clustering PSO

The cooperative data clustering PSO [13], shown in Algorithm 4, is a multi-population PSO where each population optimises part of the solution. In the case of data clustering, each population optimises a centroid position. What makes this more viable than Blackwell's [12] multi-swarm PSO is the use of a context particle. The context particle is created from the best solution of each swarm and is used in the calculation of the fitness of each particle. To calculate the fitness of entity i in subswarm j, dimension k of the context particle is replaced with entity i's solution and the fitness is calculated on the complete solution. Due to the division by zero problem described in Section III, infinity values in the results were replaced with the worst fitness encountered.

Algorithm 4 Cooperative Data Clustering PSO

```
Initialize K swarms of N one-dimensional particles each
Initialize the context particle b(k, m)
while stopping condition has not been reached do
    for each swarm S_k do    ▷ Update the context particle
        for each particle p_i in S_k do
            if fitness(b(k, S_k.x_i)) < fitness(b(k, S_k.y_i)) then
                S_k.y_i = S_k.x_i
            end if
            if fitness(b(k, S_k.y_i)) < fitness(b(k, S_k.ŷ_i)) then
                S_k.ŷ_i = S_k.y_i
            end if
        end for
        ▷ Update particles using the new context particle
        for each particle p_i in S_k do
            Update velocity using equation (1)
            Update position using equation (2)
        end for
    end for
end while
```

V. Cooperative-Multipopulation Data Clustering PSO

The cooperative-multipopulation data clustering PSO was developed in an attempt to adapt Blackwell's multi-swarm approach to the clustering problem, by making each population optimise only one centroid rather than all the centroids in parallel. The cooperative-multipopulation uses K swarms, each optimising a different centroid, as shown in Figure 2, where the best solution of each swarm is one centroid position and the solution to the optimisation problem is the combination of each swarm's global best particle. The adaptation of Blackwell's [12] approach evaluated in this paper consists of various swarms each optimising all centroids at once, as shown in Figure 3, where the best particle of each swarm consists of all the centroids and the particle with the best fitness among all swarm's best particles is selected as the solution to the optimisation problem. In both figures an example of the best solution has been marked by surrounding it with a rectangle. By separating the centroids into different swarms, the quantization error of each complete solution can not be calculated without introducing some form of communication between the swarms. This communication was introduced by adding a context particle, exactly like the one described in the cooperative PSO. What makes this algorithm different to the cooperative PSO is the repulsion and anti-convergence mechanisms borrowed from the Blackwell's multi-swarm PSO.
The main two problems with PSOs in dynamic environments [12] are the loss of diversity of the swarm and outdated memory of the individuals of the swarm. The loss of diversity

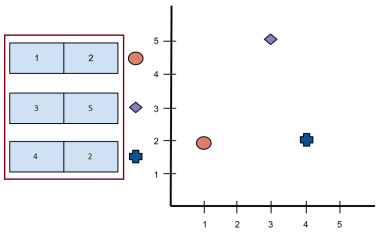


Fig. 2: Example of centroid positions of global best particles from three swarms in a Cooperative-multipopulation PSO

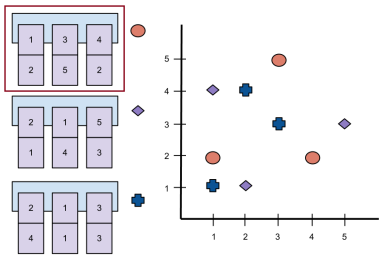


Fig. 3: Example of centroid positions of global best particles from three swarms in a Multi-swarm PSO

refers to the fact that PSOs attempt to converge around a good solution. If a change occurs within the environment, the solution towards which particles are moving may not be the best solution any more and the PSO may struggle due to the lack of exploration that takes place in the later stages of the algorithm. If particles are not diverse enough, not enough exploration occurs and the PSO can become stuck in a local optimum. The second problem, outdated memory, refers to the global best and personal best values found by particles no longer being applicable once the environment has changed.

Blackwell's [12] multi-swarm approach aims at remedying these problems in two ways. If all swarms are converging, the weakest one is re-initialised in order to inject diversity into the algorithm. There is, however, also a chance that swarms attempt to converge around the same solution. To avoid this, Blackwell [12] used repulsion. Here, if two swarm's global bests are within a certain distance from each other, the swarm with the better fitness value is re-initialised.

The cooperative-multipopulation data clustering PSO, shown in Algorithm 5, initialises K populations, where K is the number of cluster centroids to be optimised. Each population is assigned the task of finding an optimal cluster centroid position. A context particle holding all centroid positions is generated using global best solutions from each population, regardless of whether the new context particle will have a better fitness than the previous one. This context particle is used in order to calculate the fitness of a particle. As the quantization error fitness measure requires information about all clusters, the context particle holds the information about all clusters. When a particle's fitness needs to be calculated, the centroid position that the particle holds is used in combination with the centroid positions held by the context particle, just like in the cooperative PSO.

Each swarm performs the standard data clustering PSO, while the multi-population PSO repulsion and anti-convergence mechanisms are applied to all the swarms. Swarms that are optimising the same centroid position are repelled and one swarm is always exploring the search space by re-initialising a swarm when all swarms are converging.

The context particle is a representation of the global best centroids found by each swarm. This means that the final solution to the optimisation problem is this context particle. The anti-convergence introduced by the multi-swarm strategy, however, ensures that as soon as all swarms begin to converge, one is re-initialised. This means that one of the centroids of the context particle, once swarms have converged, is a re-initialised centroid, making it an inaccurate representation of the solution. To avoid this problem an extra swarm, referred to as the "explorer swarm" was introduced. Once a swarm is re-initialised, it gains the status of "explorer swarm" and the previous "explorer swarm" takes its place. This ensures that the context particle only holds centroid positions which have been exploited by the algorithm, and not random positions caused by the re-initialisation of a swarm.

Algorithm 5 Cooperative-MultiPopulation PSO

```
Initialize K + 1 swarms
for each swarm S_k do
    initialize particles
end for
Initialize the context particle b(k, z)
while stopping condition is not reached do
    for each swarm S_k in Population P do
        for each particle p_i in S_k do
            if fitness(b(k, S_k.x_i)) < fitness(b(k, S_k.y_i)) then
                S_k.y_i = S_k.x_i
            end if
            if fitness(b(k, S_k.y_i)) < fitness(b(k, S_k.ŷ_i)) then
                S_k.ŷ_i = S_k.y_i
            end if
        end for
    end for
        ▷ Update particles using the new context particle
    Perform one Multi-swarm iteration, Algorithm 3
end while
```

The cooperative-multipopulation data clustering PSO described above changes the context particle to include the best solution of each population regardless of whether the fitness of the context particle itself will be better than that of the previous context particle. In order to evaluate the effect of only changing the context particle if the new context particle's value is better than the old one, an elitist version of the algorithm was developed. This elitist version only changes the context particle to include the new best solutions if the new context particle will have a better fitness than the old one, introducing additional exploitation.

VI. Experimental Setup

The datasets used were auto-generated temporal clustering datasets with 8 clusters used and described by Graaff [14]. The changes in clusters occurred by pattern migration, which means that patterns moved from one cluster to another at each interval of change. The datasets consisted of patters with either 3, 8 or 15 attributes, with changes occurring at frequencies 1 to 5 and severities 1 to 5. All datasets contained 8000 patterns and 100 timesteps. Each timestep contained 80 data patterns, and a window of size of 80 was used to slide from one timestep to the next. To determine how many iterations a timestep uses $iterationsPerTimestep = \frac{iterations}{timeStep}$ and to determine at which timestep a change occurs, $timeStepOfChange = \frac{f}{10} * timeStep$. Therefore, to determine at which iteration a change occurs use $timeStepOfChange * iterationsPerTimestep$ which results in $iterationOfChange = \frac{f}{10} * totalIterations$. Note that a "higher frequency of change" in this paper therefore means that less changes occur in the dataset.

Particles were initialised within the bounds of the dataset. The five algorithms described were evaluated on a combination of change frequencies 1, 2, 3, 4 and 5 (where a lower frequency means more changes occur), change severities 1, 2, 3, 4 and 5 (where a higher severity means a larger change) and dimensions 3, 8 and 15. The results of each experiment were averaged over 30 independent runs and the algorithms ran for 1000 iterations with a swarm size of 50 particles. For the re-initialising algorithms 10% of the swarm was re-initialised when a change occurred. In order to keep centroids within the bounds of the search space, any centroid that passed the boundaries was reset to stay on the boundary. The PSO parameters that were used are those described as good parameters by Van den Bergh [15], namely an inertia value of 0.729844, a social component of 1.496180 and a cognitive component of 1.496180.

A few measures were used to determine the clustering capabilities of the algorithm, including the inter-cluster distance, the intra-cluster distance and the Ray-turi validity index. The inter-cluster distance [14] refers to the distance between various clusters, a distance that the algorithm must maximise and which is calculated using

$$J_{iter} = \frac{2}{K(K-1)} \sum_{k=1}^{K-1} \sum_{k_2=k+1}^{K} d(\mathbf{c}_k, \mathbf{c}_{k_2}) \tag{4}$$

where K is the total number of cluster centroids, the function d is the Euclidean distance and $\mathbf{c}_k$ and $\mathbf{c}_{k_2}$ are cluster centroids at index k and k_2 respectively.

The intra-cluster distance [14] refers to the average distance of patterns from their clusters, a distance that the algorithm must minimise and which can be calculated using

$$J_{intra} = \frac{\sum_{k=1}^{K} \sum_{\forall \mathbf{z} \in C_k} d(\mathbf{z}, \mathbf{c}_k)}{|P|} \tag{5}$$

where $\mathbf{z}$ is a data pattern, C_k is a cluster, $\mathbf{c}_k$ is the cluster centroid, $|P|$ is the total number of data patterns in the dataset and d is the euclidean distance.

Lastly, the Ray-turi [14] validity index was also used in order to have a measure that combines both the intra-cluster and inter-cluster distances. Validity indexes are normally used in order to determine the optimal number of clusters for a dataset, but in this paper it was used as a more accurate representation of each algorithm's clustering performance. The Ray-turi validity index is calculated using

$$Q_{ratio} = \frac{J_{intra}}{inter_{min}} \tag{6}$$

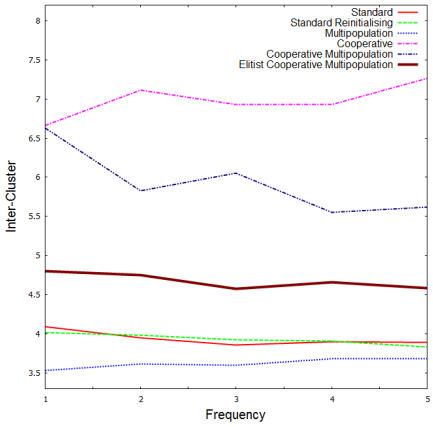
where J_{intra} is the intra-cluster distance defined in equation (5) and $inter_{min}$ is the smallest inter-cluster distance found using equation (4).

VII. Results and Discussion

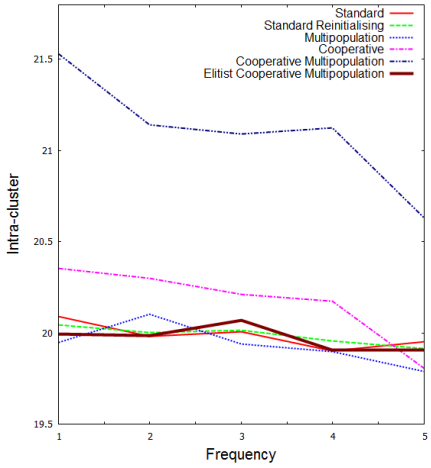
The inter-cluster, intra-cluster and Ray-Turi validity index values were used to analyse the results. These are shown for various frequencies, severities and dimensions in Figure 4 followed by the ranks of the Friedman test in Figure 5. Figure 4a and Figure 5a show the inter-cluster distance value and rank for various frequencies, where the cooperative PSO has the most optimal inter-cluster distance throughout all frequencies and it is followed by the cooperative-multipopulation PSO and its elitist variation. The same behaviour is shown in Figure 4d, Figure 5d, Figure 4g and Figure 5g which depict the values and ranks of all the algorithms' inter-cluster distances for various severities and dimensions. The graphs also show that, generally, the impact that changes in the severity and frequency values have on the inter-cluster distance is not as significant as the impact of changing the dimension, where all algorithms gained a more optimal inter-cluster distance in higher dimensions. More dimensions mean that the search space increases, allowing for the space between centroids found to be larger, which would lead to larger inter-cluster distances.

Figure 4b and Figure 5b show that there is a significant difference between the cooperative- multipopulation algorithm's intra-cluster distance to that of the rest of the algorithms. The standard PSO, re-initialising PSO, multi-swarm PSO and the elitist cooperative-multipopulation algorithms all have low values for intra-cluster distance which change by small amounts as frequencies increase. The same behaviour is shown for severity changes in Figure 4e and Figure 5e. Lastly, Figure 4h and Figure 5h show that, as dimensions increase, the elitist cooperative-multipopulation PSO improves from being the algorithm with the least optimal intra-cluster distance value to the algorithm with the most optimal value, while most other algorithms obtained a worse intra-cluster value.

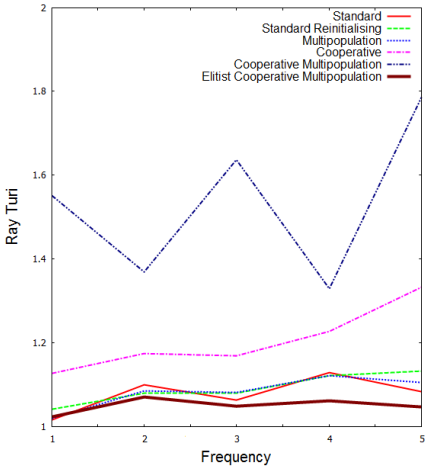
In data clustering the inter-cluster and intra-cluster distances are both important to determining the quality of a solution as clusters that are compact and separate from each other are needed. For this reason the Ray-Turi validity index was used as a measure which combines the inter-cluster and intra-cluster



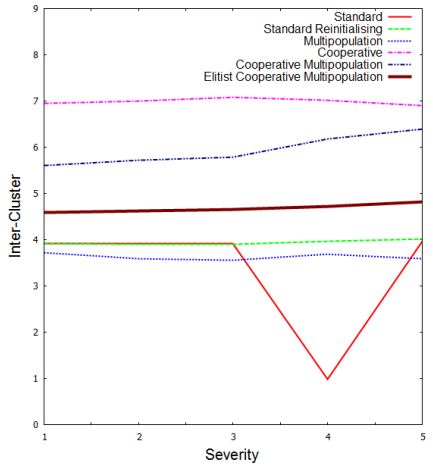
(a) Inter-cluster distance per frequency



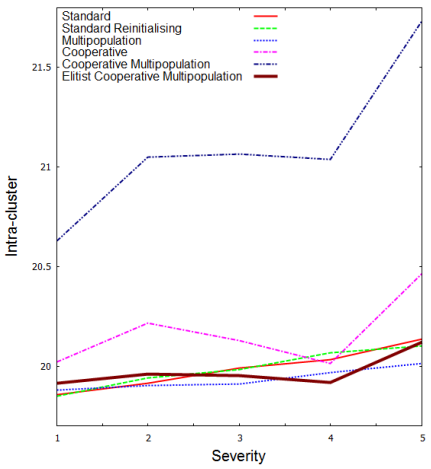
(b) Intra-cluster distance per frequency



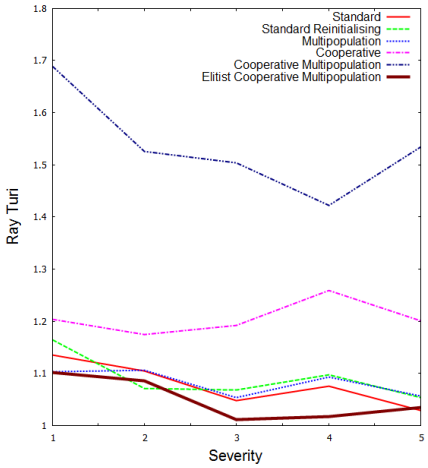
(c) Ray-turi validity per frequency



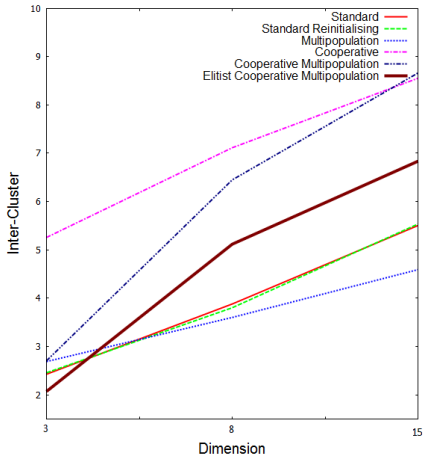
(d) Inter-cluster distance per severity



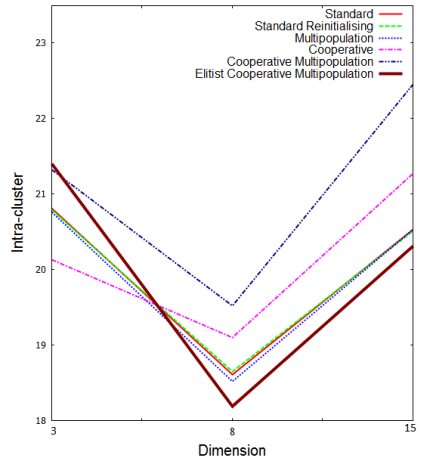
(e) Intra-cluster distance per severity



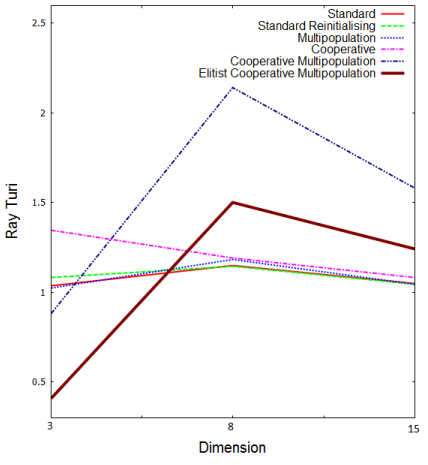
(f) Ray-turi validity per severity



(g) Inter-cluster distance per frequency

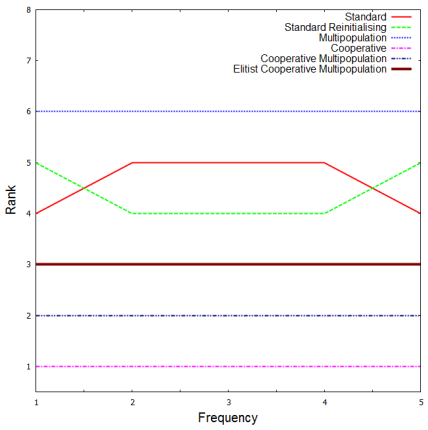


(h) Intra-cluster distance per severity

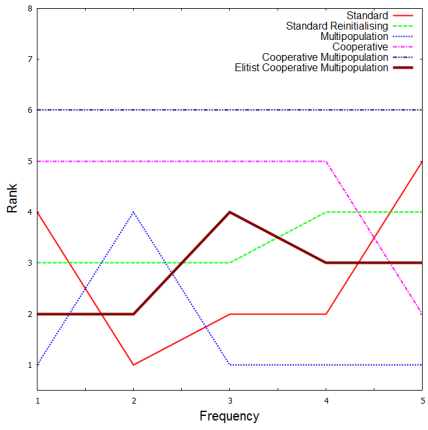


(i) Ray-turi validity per dimension

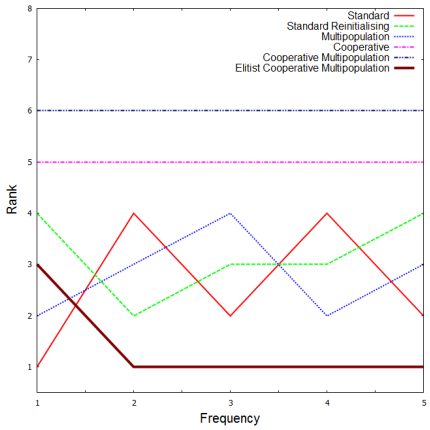
Fig. 4: Averages of Inter-cluster distance, Intra-cluster distance and Ray-turi validity index per frequency, severity and dimension



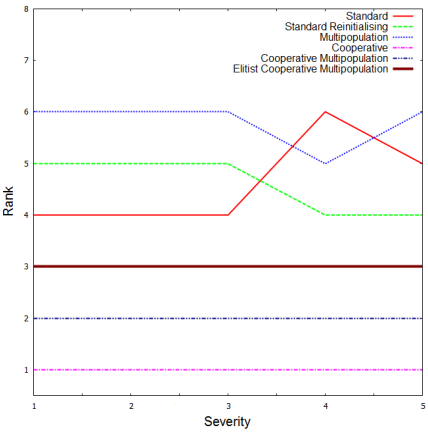
(a) Inter-cluster distance per frequency



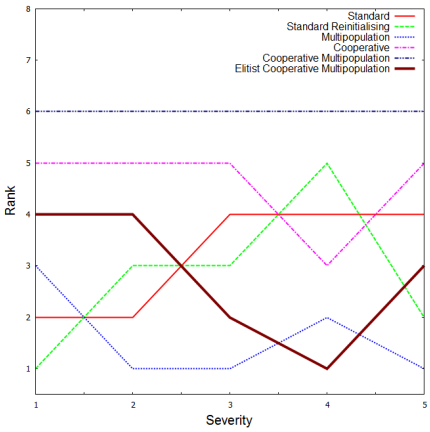
(b) Intra-cluster distance per frequency



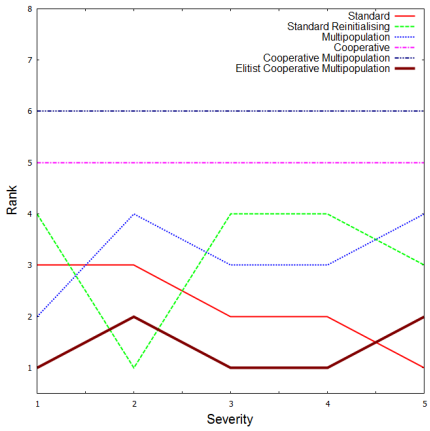
(c) Ray-turi validity per frequency



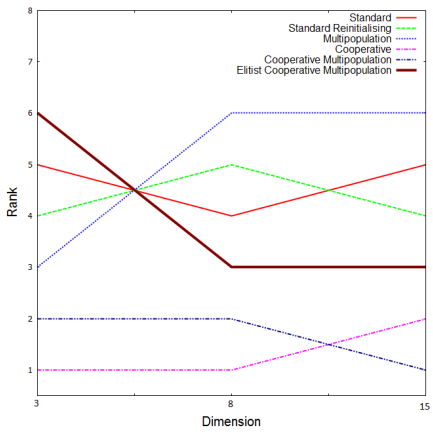
(d) Inter-cluster distance per severity



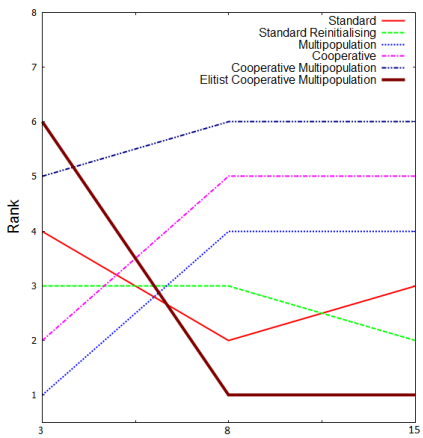
(e) Intra-cluster distance per severity



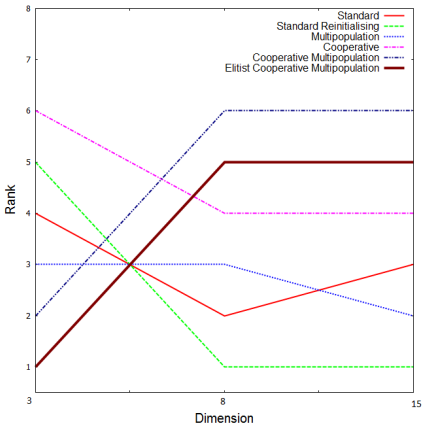
(f) Ray-turi validity per severity



(g) Inter-cluster distance per frequency



(h) Intra-cluster distance per severity



(i) Ray-turi validity per dimension

Fig. 5: Ranks of Inter-cluster distance, Intra-cluster distance and Ray-Turi validity index per frequency, severity and dimension

distances in order to give a more accurate representation of the quality of the cluster centroids found by an algorithm.

Figure 4c and Figure 5c both show the Ray-Turi validity index values for changes in frequency. The elitist cooperative-multipopulation PSO dominates with the lowest Ray-Turi validity index value throughout the majority of frequency changes. The re-initialising PSO, standard PSO and multi-swarm PSO all remain with low Ray-Turi validity index values, while the cooperative-multipopulation PSO has a significantly higher value, which is likely caused by the high intra-cluster distance shown in Figure 4b. The same behaviour can be seen in Figure 4f and Figure 5f, where, as severity values change, the elitist cooperative-multipopulation PSO remains the first or second ranked algorithm and the cooperative-multipopulation PSO remains the sixth. The algorithms displayed a different behaviour when changes in dimension took place. As shown in Figure 4i and Figure 5i the elitist cooperative- multipopulation PSO has a low Ray-Turi validity index value in low dimensions, but as the dimensions increase, so does the Ray- Turi validity value. This moves the elitist cooperative-multipopulation PSO from being ranked as the first algorithm to being ranked as the fifth, leaving the re-initialising PSO as the best ranked algorithm for higher dimensions. Lastly, the cooperative PSO improves its Ray-Turi validity index value as dimensions increase, but although this value is close to the standard PSO, re-initialising PSO and multi-swarm PSO values, it still remains ranked as the fourth algorithm.

Table I contains the average and standard deviation for inter-cluster distance, intra-cluster distance and Ray-Turi validity index for each algorithm. The table shows that the cooperative PSO has the highest inter-cluster distance, the multi-swarm PSO has the lowest intra-cluster distance and the elitist cooperative-multipopulation has the lowest Ray-Turi validity index value with a low standard deviation due to its elitist component. This confirms what was seen in Figure 4 and Figure 5 but also shows that, depending on the measure used, a different algorithm is shown as the one with the more optimal values. In order to clarify which algorithm performed the clustering task better, Figure 6 shows the actual clusters and cluster centroids for severity 5 and frequency 5 for these three algorithms at timesteps 100, 400, 700 and 1000.

Figure 6a shows that all three algorithm's best solutions are near some clusters. Figure 6b shows that by iteration 400 the multi-swarm and the elitist cooperative-multipopulation PSOs both are moving their centroids closer to the available clusters, while the cooperative PSO is moving its centroids slightly further from the clusters or leaving them in the middle of the search space. A change occurs in iteration 500 and Figure 6c shows the state of the cluster centroids at iteration 700, which is after the change. Here the multi-swarm PSO has been affected by the change and is slowly recovering from it, while the elitist cooperative-multipopulation PSO continues to move its centroids towards the correct clusters and the cooperative PSO remains with its centroids in the middle of the search space. Lastly in iteration 1000, shown in Figure 6d, the elitist cooperative-multipopulation PSO has found, or is close to finding all the clusters, while the multi-swarm and cooperative PSOs are far from the solution.

VIII. Conclusion and Future Work

This article described the implementation of six data clustering PSOs. It discussed, evaluated and compared the results of applying these PSOs to a temporal data clustering problem where patterns migrate between clusters.

It was shown that, according to the Ray-Turi validity index measure and the depiction of the cluster centroids, the elitist cooperative-multipopulation PSO performed the pattern migration data clustering task more optimally than the other algorithms but struggled with an increase in dimension. Although the inter-cluster and intra-cluster distances showed the cooperative PSO and the multi-swarm PSO to be the better choices, both measures need to be considered when determining the quality of a clustering solution.

The cooperative-multipopulation algorithm showed to be unsuccessful when considering the Ray-Turi validity index and the intra-cluster distances. This may be caused by too much re-diversification of the swarms as each re-initialisation may have been too harsh, sending the cluster centroids too far and inherently increasing the inter-cluster distance as well as the intra-cluster distance. The elitist alternative allowed for the context particle to only change if the new one would be better than the old one, allowing for the possibly harsh re-initialisations to have less of an effect.

The pattern migration problem moves data patterns from one cluster to another, leaving the clusters themselves within a certain radius of their original positions causing minor changes in the search space. This explains why the elitist cooperative- multipopulation PSO, a more exploitative algorithm, performed the clustering task more successfully than the more exploration-oriented cooperative PSO, multi-swarm PSO and cooperative-multipopulation PSO. The cooperative-multipopulation PSO is likely to perform the clustering task better than its elitist alternative in problems other than pattern migration.

Future work may include determining the best parameter combination for the cooperative-multipopulation PSO, a scalability study on the algorithms described in this paper, testing the cooperative-multipopulation PSO and its elitist alternative on problems other than pattern migration, such as the alternatives described by Graaff[14] and determining a fitness measure that does not allow for division by zero and possibly combines inter-cluster and intra-cluster distances.

References

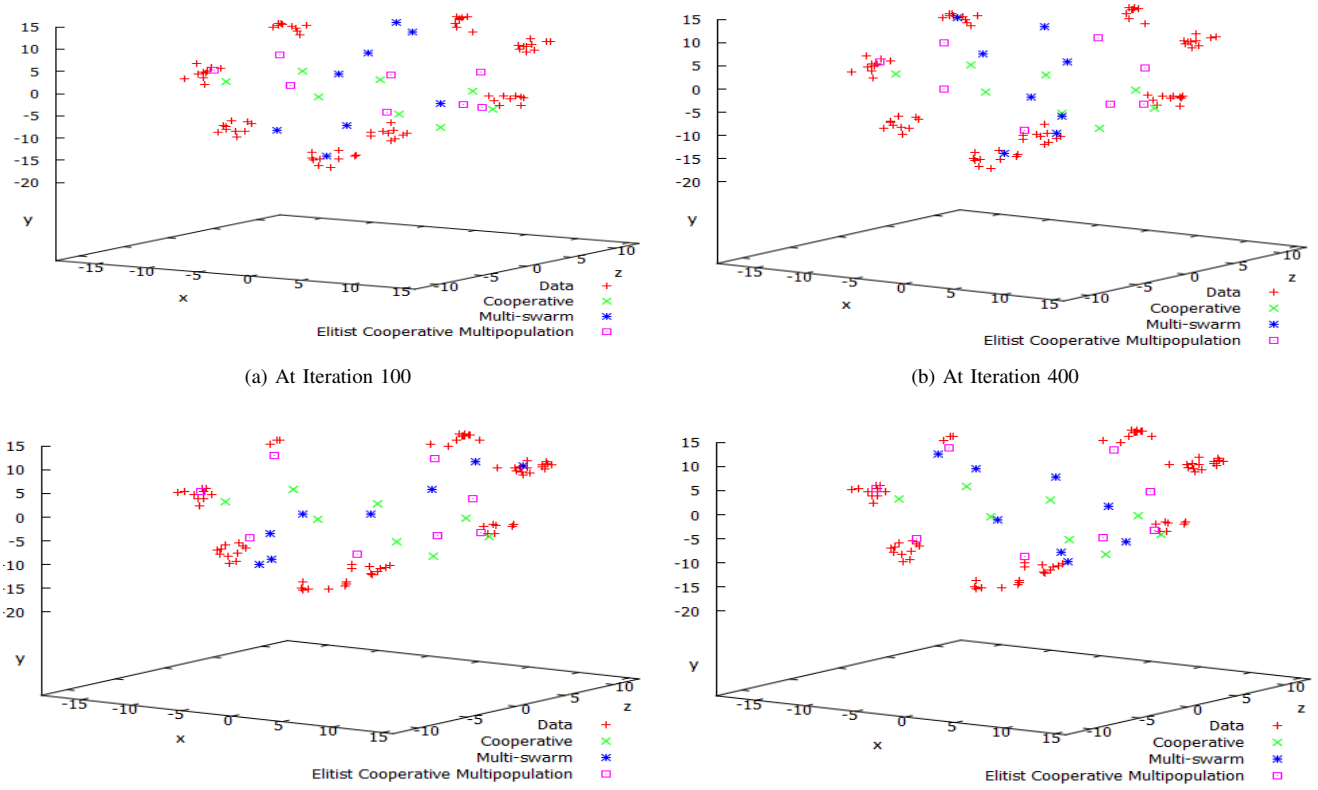
[1] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, 2009.

[2] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Fifth Berkeley Symposium on Mathematical Statististics and Probability*, vol. 1, pp. 281–297, 1967.

[3] C. S. J. W. T. C. K. Chuang, H. Tzeng, "Fuzzy c-means clustering with spatial information for image segmentation," *Computerized Medical Imaging and Graphics*, vol. 30, pp. 9–15, 2006.

TABLE I: Averages and standard deviation for inter-cluster distance, intra-cluster distance and ray-turi validity for each algorithm

Algorithm	**Inter-cluster Distance**	**Intra-cluster Distance**	**Ray-Turi Validity**
Standard	3.940 ± 1.3	19.987 ± 1.017	1.079 ± 0.116
Standard Reinitialising	3.935 ± 1.301	19.987 ± 0.995	1.091 ± 0.12
Multipopulation	3.626 ± 0.821	19.935 ± 1.054	1.083 ± 0.11
Cooperative	6.979 ± 1.593	20.168 ± 0.99	1.207 ± 0.188
Cooperative-multipopulation	5.935 ± 2.359	21.103 ± 1.275	1.535 ± 0.233
Elitist Cooperative-multipopulation	4.676 ± 2.588	19.973 ± 1.551	1.051 ± 0.682



(a) At Iteration 100

(b) At Iteration 400

(c) At Iteration 700

(d) At Iteration 1000

Fig. 6: Cluster positions for dataset of frequency 5 and severity 5 during iterations 100, 400, 700 and 1000

[4] D. S., A. A., and K. A., "Automatic clustering using an improved differential evolution algorithm," *IEEE Transactions on Systems Man and Cybernetics Part A Systems and Humans*, vol. 38, no. 1, pp. 218–237, 2008.

[5] L. T., Z. Y., H. Z., and W. Z., "A new clustering algorithm based on artificial immune system," in *Fifth International Conference on Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08*, vol. 2, pp. 347–351, 2008.

[6] H. J., V. A., and D. J., "A hierarchical unsupervised growing neural network for clustering gene expression patterns.," *Bioinformatics*, vol. 17, no. 2, pp. 126–136, 2001.

[7] D. W. V. D. Merwe and A. P. Engelbrecht, "Data clustering using particle swarm optimization," in *2003 Congress on Evolutionary Computation 2003 CEC 03 (2003)*, vol. 1, pp. 215–220, 2003.

[8] H. J. P. Zhenkui, H. Xia, "The clustering algorithm based on particle swarm optimization algorithm," in *International Conference on Intelligent Computation Technology and Automation*, vol. 1, p. 148, 2008.

[9] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Joint Conference on Neural Networks*, p. 19421948, 1995.

[10] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *IEEE Congress on Evolutionary Computation*, p. 6973, 1998.

[11] R. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *IEEE Congress on Evolutionary Computation*, vol. 1, pp. 94–100, 2001.

[12] T. Blackwell, "Particle swarm optimization in dynamic environments," *Evolutionary Computation in Dynamic and Uncertain Environments*, vol. 49, pp. 29–49, 2007.

[13] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," in *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 225–239, 2004.

[14] A. Graaff, *A Local Network Neighbourhood Artificial Immune System*. PhD thesis, University of Preotria, June 2011.

[15] F. V. den Bergh, *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, November 2001.

Dynamic Differential Evolution Algorithm for Clustering Temporal Data

Kristina S. Georgieva and Andries P. Engelbrecht

Department of Computer Science,
University of Pretoria, South Africa
kristina.s.georgieva@gmail.com
engel@cs.up.ac.za

Abstract. Temporal data clustering is the process of grouping similar patterns in a dataset together when the patterns change with time. This change in patterns introduces the issue of loss of diversity in differential evolution algorithms. The lack of re-diversification of the population limits the exploration ability of differential evolution algorithms resulting in early convergence around stale solutions. This paper describes and evaluates three algorithms that were applied to a temporal data clustering problem, namely the standard data clustering DE, the reinitialising data clustering DE, and the data clustering DynDE.

1 Introduction

Data clustering [8] refers to the process of grouping similar data patterns together, where the data can be either static or dynamic. Static data refers to already existing data which does not change, while dynamic data refers to frequently changing data [7]. Temporal data [13], which refers to data that changes at some frequency of time, can therefore be considered as a type of dynamic data due to its frequent changes.

A clustering dataset may change in various ways and combinations [5]. Patterns from one cluster may move to another cluster, clusters themselves may also move around the search space as a whole and, lastly, old clusters may disappear and new ones may appear due to migrating, disappearing or appearing patterns.

The differential evolution [16] algorithm is a population-based evolutionary algorithm that uses selection, mutation and crossover to adapt the individuals of its population. The clustering of data using differential evolution has been previously evaluated by a number of researchers [1][3][9][14][15][17].

When clustering dynamic data, the differential evolution algorithm suffers from loss of diversity [12]. This occurs as individuals begin to converge. The problem emerges when data is dynamic, as exploration of the search space is necessary to evolve to the correct positions and the population needs to be re-diversified in order to explore the search space. Unlike particle swarm optimization algorithms, however, DEs do not have the issue of outdated memory [2] as they do not store any information about the population's previous state. This may make them more appropriate algorithms for clustering dynamic data.

This paper evaluates the temporal data clustering ability of the data clustering DE proposed in [6]. The paper also adapts this algorithm to re-initialise part of the population if a change in the data has occurred. Lastly, it adapts the DynDE proposed in [10] to perform the clustering process. The three DEs are compared using three measures [5], namely the inter-cluster distance, the intra-cluster distance and the Ray-Turi validity index. These results are then ranked using the Friedman test.

2 Differential Evolution

The differential evolution [16] algorithm is a population-based evolutionary algorithm guided by distance information about the current population [4]. Each possible solution in the population is a vector, called an individual. Individuals adapt using three evolutionary processes, namely selection, mutation and crossover, where the mutation process is the one that mainly differentiates the DE from all other evolutionary algorithms.

The mutation process uses a target vector selected from the population at random, as well as distance information from the population to generate a new vector called the trial vector. This is done by adding a scaled difference between two random individuals of the population to a target vector selected at random. The trial vector is created using [4]

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)) \tag{1}$$

$\mathbf{x}_{i_1}(t)$ is the target vector, β is the scaling factor, $\mathbf{x}_{i_2}(t)$ is a randomly selected individual from the population that is not $\mathbf{x}_{i_1}(t)$ and $\mathbf{x}_{i_3}(t)$ is a randomly selected individual from the population that is not $\mathbf{x}_{i_1}(t)$ or $\mathbf{x}_{i_2}(t)$.

The current individual being adapted, $\mathbf{x}_i$, is then used by the crossover process to produce a single offspring. The offspring is created by [4]

$$x'_{ij}(t) = \begin{cases} u_{ij} & \text{if } j \text{ is } \in J \\ x_{ij}(t) & \text{if } j \text{ is } \notin J \end{cases} \tag{2}$$

where $\mathbf{u}_i$ is the trial vector, $\mathbf{x}_i$ is the parent vector, and J is a set of crossover points, which are determined by random selection. This paper uses binomial crossover where a crossover probability is used to determine which dimensions of the offspring will become part of the new solutions and which dimensions of the parent vector will remain as part of the solution, ensuring that one randomly selected dimension is forced to be within the set J.

Lastly, a selection algorithm is applied in order to determine which offspring and which parents survive to the next generation. This paper uses an elitist selection strategy, where the individual with the best fitness value between the offspring and the parent survives to the next generation.

3 Data Clustering DE and its reinitialising alternative

The solution to a data clustering problem consists of K vectors, where K is the total number of clusters and each vector represents the position of the center of

one cluster. Due to the solution no longer being one vector, the representation of individuals in a clustering DE needs to change. The clustering DE, therefore, represents each solution as a set of centroid positions, where each dimension of the solution is a vector holding the position of a cluster centroid.

The dataset has an influence on determining whether one solution is better or worse than another solution. For this reason, the dataset needs to be used when the fitness of an individual is calculated. This is done by using the quantization error [11] as a fitness measure for the clustering problem. The quantization error is minimised and it is calculated using

$$J_e = \frac{\sum_{k=1}^{K} \frac{\sum_{\forall \mathbf{z_p} \in C_k} d(\mathbf{z_p}, \mathbf{c_k})}{|C_k|}}{K} \tag{3}$$

where $\mathbf{z}$ is a data pattern, C_k is the set of data patterns assigned to the cluster, $d(\mathbf{z_p}, \mathbf{c_k})$ is the Euclidean distance between data pattern $\mathbf{z_p}$ and cluster centroid $\mathbf{c_k}$, $|C_k|$ is the total number of patterns assigned to the centroid, and K is the total number of centroids. This measure can divide by zero if a centroid is positioned far from all patterns leading to no patterns being assigned to it, making $|C_k|$ zero. To remedy this, if $|C_k|$ was zero, the fitness value was approximated to infinity, such bad fitness will, with time, filter out the bad solution.

The data clustering DE [6] performs the selection, mutation and crossover processes in the same manner as the DE described in section 2. A target vector and two difference vectors are selected at random, a trial vector is then created using equation(1) on each cluster centroid position, the offspring and the parent individual are then crossed over using equation(2) on each cluster centroid, and then the individual with the best fitness is selected to survive to the next generation. The complete algorithm is shown in Algorithm 1.

The reinitialising data clustering DE adapts the data clustering DE in an attempt to re-diversify the population and overcome the loss of diversity problem that the DE experiences. This adaptation is the addition of a re-initialisation step, where part of the population is re-initialised if a change in the dataset has been detected.

4 DynDE for Data Clustering

The DynDE algorithm proposed by Mendes and Mohais [10] adapts the original DE described in section 3 by adding an element of exclusion and Brownian individuals in order to increase diversity. It is a multi-population algorithm where each population searches for the optimal solution to a problem in a different area of the search space. If two populations are optimising the same area of the search space, the population with the worst global best solution is re-initialised.

Brownian individuals are individuals that are not adapted using the standard DE strategies discussed, but instead their new positions are generated based on the global best position of their population and Gaussian noise. Gaussian noise is added to the position of the global best individual of a sub-swarm using

$\mathbf{b}_i = \mathbf{y}_i + \mathbf{N}(0, \sigma)$ where $\mathbf{b}_i$ is the new value for dimension i of the Brownian individual, $\mathbf{y}_i$ the value for dimension i of the global best individual and $\mathbf{N}(0, \sigma)$ is Gaussian noise with mean 0 and standard deviation σ. Mendes and Mohais [10] evaluated quantum individuals and entropic differential evolution, but concluded that the Brownian individuals displayed the most successful results.

Algorithm 1 Data Clustering Differential Evolution

1: Initialize a population of N individuals with K centroid positions each
2: **while** stopping condition has not been reached **do**
3: **for** each individual x_i **do**
4: **for** each data pattern $\mathbf{z_p}$ **do**
5: Calculate the Euclidean distance $d(\mathbf{z_p}, \mathbf{x_{ij}})$ to all clusters
6: Assign pattern $\mathbf{z_p}$ to cluster C_{ij}, such that $d(\mathbf{z_p}, \mathbf{x_{ij}}) = min_{\forall k=1...N_k} \{d(\mathbf{z_p}, \mathbf{x_{ik}})\}$
7: **end for**
8: calculate the fitness using equation (3)
9: Select a target entity and two difference entities
10: Generate a trial entity $u_i(t)$ using equation (1)
11: Generate the offspring x'_i using equation (2)
12: **if** $f(x'_i)$ is better than $f(x_i)$ **then**
13: add x'_i to the next population
14: **else**
15: add x_i to the next population
16: **end if**
17: **end for**
18: **end while**

Algorithm 2 DynDE

1: Initialise M populations of individuals
2: Assign a percentage of each population to be Brownian Individuals
3: Evaluate each population by performing the steps described in lines 4-9 of Algorithm 1
4: Compare the global best individuals from each population to each other
5: **if** One or more centroids of best individuals of two populations are within an exclusion radius r_{excl} of each other **then**
6: re-initialise the population with the worst global best
7: **else**
8: **for** each population **do**
9: **for** each individual x_i **do**
10: **if** x_i is a normal individual **then**
11: Update x_i using lines 10-12 of Algorithm 1
12: **else if** x_i is a Brownian Individual **then**
13: Update using $b_i = y_i + N(0, \sigma)$
14: **end if**
15: **end for**
16: **end for**
17: Make the old Brownian Individuals normal Individuals
18: Make a percentage of the weakest individuals Brownian Individuals
19: **end if**

The DynDE algorithm begins by initialising the sub-populations, making a percentage of each sub-population Brownian individuals. It then calculates the fitness of all the individuals in each sub-population. If two populations are within an exclusion radius r_{excl} from each other, the entire population with the worst global best solution is re-initialised to random positions within the search space.

The populations that were not re-initialised update their normal individuals by using the standard DE update methods described in section 2 and their Brownian individuals using the Brownian individual update. For this paper an asynchronous approach was taken when updating the DynDE individuals, where, instead of adding survivors to the next population, survivors replace their parents in the current population.

What changes when the DynDE algorithm is used for data clustering is simple. The representation of individuals described for the data clustering DE is used and the fitness calculation takes the dataset into consideration. Inherently, the data clustering DE updates are used instead of the DE updates described in section 3. This algorithm is shown in Algorithm 2, where the number of clusters is known.

5 Experimental Procedure

The migrating patterns cluster datasets used in [5] were used as the clustering problems in this paper. Datasets had 8 clusters, 80 patterns per timestep, 100 timesteps and combinations of severities of 1,2,3,4 and 5, frequencies of 1,2,3,4 and 5, and dimensions 3,8 and 15. A window of 80 patterns was used to slide from one timestep to the next. Higher severities refer to larger changes while higher frequencies refer to less intervals of change, where the $iterationOfChange = \frac{f}{10} * totalIterations$.

Fifty individuals were used and were initialised within the bounds of the dataset. The averages of 30 individual simulations are reported in this paper, where each simulation ran for 1000 generations with 8 populations. For the reinitialising DE, 10% of the population was re-initialised when a change took place. A boundary constraint was also used, where individuals that pass the boundaries were reset to stay on the boundary. Lastly, a scaling factor and crossover probability of 0.5 were used.

6 Results and Discussion

The inter-cluster distance, intra-cluster distance and Ray-Turi validity index were used as measures to compare the clustering ability of each of the three DEs. Table 1 shows the average values and standard deviations for each of these three measures. On average, the standard and reinitialising data clustering DEs have the best inter-cluster distances, showing that the clusters found by these algorithms are further apart than the ones found by the DynDE. The DynDE, however, has a more optimal average intra-cluster distance, showing that the clusters found by the algorithm were more compact. Lastly, the DynDE's Ray-Turi validity value shows that the overall clustering capability of the DynDE surpassed that of the standard and reinitialising DEs.

Table 1: Averages, standard deviation and chi-square values for inter-cluster distance, intra-cluster distance and Ray-Turi validity for each algorithm's last iteration. Where the Chi-square was calculated with $n = 75, k = 3, df = 2, \alpha = 95\%$, showing a statistical significant difference for all three measures where $\chi^2 > 5.991$

Algorithm	**Inter-cluster Distance**	**Intra-cluster Distance**	**Ray-Turi Validity**
Standard	9.029 ± 2.700	21.769 ± 1.191	1.084 ± 0.166
Standard Reinitialising	9.020 ± 2.688	21.772 ± 1.188	1.0716 ± 0.168
DynDE	3.262 ± 1.214	20.074 ± 1.316	0.902 ± 0.388
Chi-Square value χ^2	112.6666667	21.94666667	10.90666667

Figure 1 illustrates the ranks resulting from the Friedman test performed for the severity, frequency and dimension changes. Figures 1a, 1b, 1d, 1e, 1g and 1h show that the DynDE has the last-ranking inter-cluster distance, but the best intra-cluster distance for all frequency and severity changes and most dimension changes. The Ray-Turi validity value takes into account both inter- and intra-cluster distances giving a more accurate representation of the clustering ability of an algorithm. As shown in figures 1c and 1f, DynDE has the first-ranking Ray-Turi validity value for all frequency and severity changes. Figure 1i, on the other hand, shows that the DynDE struggled with higher dimensions by displaying the worst Ray-Turi validity value out of the three algorithms as the dimension increased. Overall the DynDE algorithm received the best resulting intra-cluster distance and Ray-Turi validity values, while it received the worst inter-cluster distance values.

7 Conclusion and Future Work

This paper evaluated and compared the temporal clustering abilities of three data clustering DEs. These three algorithms involve the standard data clustering DE, the reinitialising data clustering DE, and DynDE.

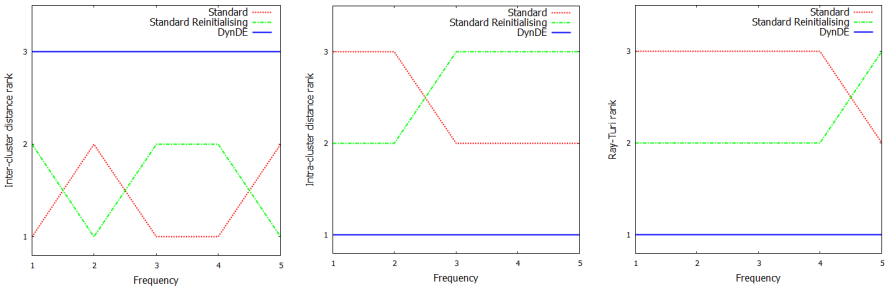
The DynDE's good intra-cluster distance showed that the DynDE found clusters where the data patterns are close to each other, while the bad inter-cluster distance shows that the clusters found were not as far from each other as the ones found by the other two algorithms. According to the Ray-Turi validity value, which combines the inter- and intra-cluster distances, the DynDE algorithm performs the temporal data clustering tasks more effectively than the standard and reinitialising data clustering DEs. The addition of repulsion in order to re-diversify populations and the addition of Brownian individuals to exploit good solutions had a positive effect on the performance of differential evolution for clustering temporal data.

Future work includes implementing and evaluating a DynDE clustering solution where each population optimises one optimum instead of all, an algorithm

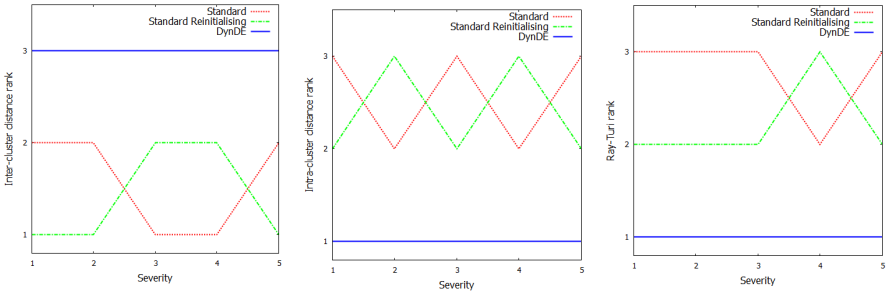
that does not require prior knowledge of the number of clusters and, lastly, parameter tuning for these clustering DEs. Improvements to this algorithm in order to make it less sensitive to dimension changes can also be considered for future work.

References

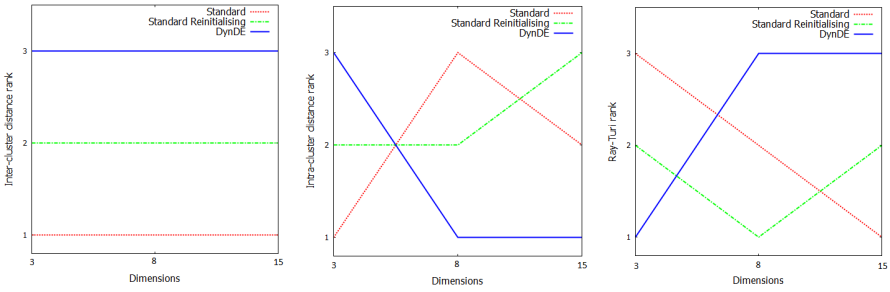
1. Abraham, A.; Das, S., Konar, A.: Document clustering using differential evolution. In: Congress on Evolutionary Computation. pp. 1784–1791 (2006)
2. Blackwell, T.: Particle swarm optimization in dynamic environments. Evolutionary Computation in Dynamic and Uncertain Environments pp. 29–49 (2007)
3. Das, S.; Abraham, A., Konar, A.: Automatic clustering using an improved differential evolution algorithm. IEEE Transactions on Systems Man and Cybernetics Part A Systems and Humans 38(1), 218–237 (2008)
4. Engelbrecht, A.P.: Computational Intelligence: An Introduction, chap. 1. Wiley Publishing (2007)
5. Graaff, A.: A Local Network Neighbourhood Artificial Immune System. Ph.D. thesis, University of Preotria (June 2011)
6. Hanuman, S.; Babu, V., Govardhan, A., Satapathy, S.: Data clustering using almost parameter free differential evolution technique. International Journal of Computer Applications 8(13) (October 2010)
7. Iyengar, A.; Ramaswamy, L., Schroeder, B.: Web Information Systems Engineering and Internet Technologies Book Series, vol. 2, chap. Chapter 5, pp. 101–130. Springer (2005)
8. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern Recognition Letters (2009)
9. Maulik, U.; Saha, I.: Modified differential evolution based fuzzy clustering for pixel classification in remote sensing imaging. Pattern Recognition 42(9), 2135–2149 (2009)
10. Mendes, R.; Mohais, A.S.: Dynde: a differential evolution for dynamic optimization problems. In: Evolutionary Computation, 2005. The 2005 IEEE Congress on. vol. 3, pp. 2808 – 2815 Vol. 3 (sept 2005)
11. van der Merwe, D.W.; Engelbrecht, A.: Data clustering using particle swarm optimization. In: Evolutionary Computation, 2003. CEC '03. The 2003 Congress on. vol. 1, pp. 215 – 220 Vol.1 (dec 2003)
12. Pant, M.; Thangaraj, R., Abraham, A., Grosan, C.: Differential evolution with laplace mutation operator. In: Congress on Evolutionary Computation. pp. 2841–2849 (2009)
13. Patel, J.: Temporal database system. Master's thesis, Department of Computing, Imperial College, University of London, (June 2003)
14. Paterlini, S.; Krink, T.: High performance clustering with differential evolution. In: Congress on Evolutionary Computation. vol. 2, pp. 2004–2011 (2004)
15. Paterlini, S.; Krink, T.: Differential evolution and particle swarm optimisation in partitional clustering. Computational Statistics & Data Analysis 50(5), 1200–1247 (2006)
16. Price, K.; Storn, R., Lampinen, J.: Differential evolution: A practical approach to global optimization. Springer (2005)
17. Tian, Y.; Liu, D., Qi, H.: K-harmonic means data clustering with differential evolution. In: Conference on Future BioMedical Information Engineering. pp. 369–372 (2009)



(a) Inter-cluster distance per frequency (b) Intra-cluster distance per frequency (c) Ray-turi validity per frequency



(d) Inter-cluster distance per severity (e) Intra-cluster distance per severity (f) Ray-turi validity per severity



(g) Inter-cluster distance per frequency (h) Intra-cluster distance per severity (i) Ray-turi validity per dimension

Fig. 1: Ranks of Inter-cluster distance, Intra-cluster distance and Ray-Turi validity index per frequency, severity and dimension

Cooperative DynDE for Temporal Data Clustering

Kristina S. Georgieva and Andries P. Engelbrecht

***Abstract*— Temporal data is common in real-world datasets. Clustering of such data allows for relationships between data patterns over time to be discovered. Differential evolution (DE) algorithms have previously been used to cluster temporal data. This paper proposes the cooperative data clustering dynamic DE algorithm (CDCDynDE), which is an adaptation to the data clustering dynamic DE (DCDynDE) algorithm where each population searches for a single cluster centroid. The paper applies the proposed algorithm to a variety of temporal datasets with different frequencies of change, severities of change, dataset dimensions and data migration types. The clustering results of the cooperative data clustering DynDE are compared against the original data clustering DynDE, the re-initialising data clustering DE and the standard data clustering DE. A statistical analysis of these results shows that the cooperative data clustering DynDE algorithm obtains better data clustering solutions to the other three algorithms despite changes in frequency, severity, dimension and data migration types.**

I. Introduction

Data clustering is a task that aims to determine relationships among objects by grouping similar objects together [1]. This task has been previously performed with a variety of computational intelligence (CI) algorithms, including neural networks [2], swarm intelligence algorithms [3][4][5], artificial immune systems [6][7] and evolutionary algorithms [8][9].

Temporal data [10], which is data that changes with time, introduces additional complexities for data clustering algorithms. The changing data results in the clustering solution differing from one time-step to another, requiring the algorithms to track the new centroids. Changes in the environment, which refer to a change in the constitution of the clusters within that environment, may include clusters becoming larger, smaller, appearing, disappearing, moving, and patterns that migrate to other clusters. These changes make it difficult for algorithms to cluster data due to the loss of diversity and outdated memory [11] suffered by the population-based algorithms. The loss of diversity refers to individuals in the population of an algorithm converging to a solution, making the population less diverse. This phenomenon is problematic in temporal environments as few or no individuals explore the search space for better solutions when the environment changes. Outdated memory refers to the optimal values found by an algorithm's population becoming obsolete as the environment changes.

Few computational intelligence algorithms for clustering temporal data have previously been implemented. Examples of such algorithms include the local network neighbourhood artificial immune system [6], re-initialising data clustering PSO [12], multi-population data-clustering PSO [12], cooperative-multipopulation data-clustering PSO [12], re-initialising data clustering DE [13], data clustering dynamic DE (DCDynDE) [13] and a combination of self-organising maps (SOMs) and autoregressive integrated moving average (ARIMA) time-series models [14].

This paper focuses on differential evolution (DE) approaches to cluster temporal data. Georgieva and Engelbrecht [13] proposed two adaptations to Hanuman et al's [9] data clustering DE in order to handle temporal data. The first algorithm proposed was the re-initialising data clustering DE (RDCDE), which re-initialises part of a population when a change occurs. The second algorithm proposed was the data clustering dynamic DE (DCDynDE), which uses repulsion and Brownian individuals in order to re-diversify the population. This paper proposes the cooperative data clustering DynDE (CDCDynDE), which adapts the DCDynDE by making each population search for a single optimum instead of for all optimums in parallel to the other populations. This allows for various areas of the search space to be exploited for clusters, while the repulsion and Brownian individuals promote exploration in the temporal environment.

An empirical analysis of the CDCDynDE in comparison with Hanuman et al's [9] data clustering DE, the RDCDE and the DCDynDE is done in this paper. The results show that the CDCDynDE performed the clustering task more effectively than the other algorithms. The algorithm's performance remained good despite changes in frequency, severity, dimension and pattern migration types.

The article begins with Section II summarises the DE, Hanuman et al's [9] data clustering DE, RDCDE and DynDE in Section. These summaries are followed by the proposed CDCDynDE algorithm in Section III, the experimental set-up in Section IV and the results of the performed experiments in Section V. Lastly, Section VI summarises the article's findings along with possible future work.

II. Differential Evolution

The DE is a population-based evolutionary algorithm (EA) that has been used to solve a wide range of optimization problems [13][15][16][17]. The DE, like other EAs makes use of evolutionary operators to adapt individuals. These operators are mutation, crossover, and selection. DE differs from other EAs by applying these operators in a different order, as well as introducing diversity via the mutation of each parent individual and producing a trial vector. Mutational step sizes are based on distances that randomly selected individuals are from one another. The trial vector is recombined with the parent individual, using a discrete crossover operator, to produce a single offspring. The offspring then competes with the parent individual, through a selection process, to survive to the next generation.

The mutation operator creates a trial vector for each parent individual $\mathbf{x}_i$ using

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)) \quad (1)$$

where $\mathbf{x}_{i_1}(t)$ is the target vector, which in this paper is a randomly selected individual, $\mathbf{x}_{i_2}(t)$ and $\mathbf{x}_{i_3}(t)$ are two randomly selected individuals used to create a difference vector with $i_1 \neq i_2 \neq i_3$, and β is a constant, called the scaling factor.

The crossover operator implements a discrete recombination of the parent individual and the trial vector using

$$\mathbf{x}'_{ij}(t) = \begin{cases} \mathbf{u}_{ij} & \text{if } j \text{ is } \in J \\ \mathbf{x}_{ij}(t) & \text{if } j \text{ is } \notin J \end{cases} \quad (2)$$

where $\mathbf{u}_i$ is the trial vector and J is a set of crossover points. The crossover points are determined differently depending on whether exponential or binomial crossover is used. For binomial crossover a crossover probability is used to randomly determine which points are to be crossed over. For exponential crossover an index is randomly selected and a sequence of adjacent indexes are chosen to be crossed over. The algorithms described in this paper use binomial crossover.

An elitist selection operator is used to accept the best individual between the parent and offspring to survive to the next generation.

DE has been developed to solve continuous-valued optimization problems, and has been shown to be very efficient [18][19][20][21]. Data clustering can be defined as an optimization problem, where the objective is to find cluster centroids that maximise the compactness and separation of clusters. Since data clustering is in effect an optimization problem, DE can be applied to find optimal cluster centroids.

Hanuman et al [9] adapted the DE in order to solve data clustering problems, referred to in this paper as the standard data clustering DE (SDCDE) algorithm. The first change to the algorithm is the representation of an individual. In the DE algorithm an individual is represented as an n-dimensional vector, where n is the dimensionality of the problem. The solution to a clustering problem, however, is a group of cluster centroids. Each cluster centroid is an n_x-dimensional vector. Therefore the dimension of a SDCDE individual is $n = n_x * n_K$, where n_K is the number of cluster centroids.

The second change to the DE algorithm is the use of the dataset to guide the search. Before the fitness of an individual is calculated, each data pattern is assigned to the cluster centroid closest to it, which is determined using the Euclidean distance measure. The fitness is then calculated using the distances between data patterns and the centroids that the patterns are assigned to. This paper uses the quantization error as the fitness measure, which is calculated using [22]

$$J_e = \begin{cases} \frac{\sum_{k=1}^{n_K} \frac{\sum_{\forall \mathbf{z}_p \in C_k} d(\mathbf{z}_p, \mathbf{c}_k)}{|C_k|}}{n_K} & \text{if } |C_k| > 0 \\ \infty & \text{if } |C_k| = 0 \end{cases} \quad (3)$$

where $\mathbf{z}_p$ is a data pattern, C_k is a cluster, $\mathbf{c}_k$ is the cluster centroid of C_k and $|C_k|$ is the total number of data patterns in cluster C_k.

DE algorithms suffer from loss of diversity in temporal environments [23], rendering it inefficient to cluster temporal data unless a mechanism is incorporated to re-diversify the population. Georgieva and Engelbrecht [13] proposed two adaptations to the SDCDE:

- The re-initialising data clustering DE (RDCDE) adapts the DCDE by re-initialising a percentage of the population if a change in the environment occurs. The re-initialisation places a percentage of the population at randomly selected positions in the search space, allowing re-initialised individuals to explore the environment for new solutions.
- The data clustering dynamic DE (DCDynDE) combines the DCDE with Mendes and Mohais' [24] DynDE algorithm. The DynDE is a multi-population DE that adds Brownian individuals and exclusion to the DE in order to increase the diversity of the population. The DynDE selects a percentage of the weakest individuals in a sub-population to become Brownian individuals during each iteration. Brownian individuals are individuals generated by adding Gaussian noise to the position of best individual of a sub-population. Additionally, if two sub-populations are within a certain radius of each other, the weakest sub-population is excluded by being re-initialised.
  The DCDynDE adapts the DynDE by changing the representation of the individuals to contain a group of centroid positions. Additionally, the data patterns are assigned to their closest centroid and the fitness of each solution is determined using the quantization error defined in Equation (3).

The datasets used are automatically generated datasets with known characteristics. The time-step when a change occurs is, therefore, priorly known and was provided to the RDCDE. The DCDynDE, on the other hand, is capable of tracking changes in the environment through repulsion and Brownian individuals, requiring no prior knowledge of the time-step when a change occurs.

III. Cooperative Data Clustering Dynamic Differential Evolution

The DCDynDE [13] algorithm consists of multiple populations working in parallel, where the final solution is the best global best over all sub-populations. This section proposes an adaptation of the DCDynDE, by letting each sub-population search for a single centroid instead of all of the centroids. By doing this, the task of each sub-population is reduced to finding an n_x-dimensional vector instead of an $n_x * n_K$-dimensional solution, focusing the search. This approach is similar to the cooperative PSO developed by Van den Bergh and Engelbrecht [25].

The CDCDynDE algorithm initialises n_K sub-populations of n_x-dimensional individuals, where n_K is the total number

of clusters. The quantization error requires knowledge of all cluster centroid positions in order to determine the suitability of a clustering solution. By making each population search for a single centroid position, this knowledge is lost. In order to calculate the fitness of an individual, CDCDynDE introduces a context individual, inspired by the cooperative PSO [25]. A context individual is a $n_x * n_K$-dimensional individual comprised of the best n_x-dimensional centroids from each sub-population.

To calculate the fitness of a solution belonging to sub-population S_k, the dimension k of the context individual is replaced with that solution. The remainder of the context individual consists of the best solutions of the other populations. The fitness of this adapted context individual is then calculated and assigned as the fitness of the solution.

During each iteration of the CDCDynDE the distance between the global best solutions of each sub-population is calculated. If the distance between two sub-populations is smaller than a pre-defined exclusion radius, r_{excl}, the weakest of the two sub-populations is re-initialised. The re-initialisation involves placing all individuals within the sub-population at random positions in the search space. The re-initialised sub-population therefore restarts its search for an optimal centroid. This exclusion mechanism, taken from the DCDynDE algorithm, allows for each sub-population to find a different centroid and promotes separability between clusters.

After the exclusion has taken place, the sub-populations that were not re-initialised are adapted using the standard mutation, crossover and selection DE mechanisms. A percentage, $b_{perc}\%$, of the weakest individuals in each sub-population are then selected to become Brownian individuals. These Brownian individuals are adapted using

$$w_{ij} = y_{ij} + N(0, \sigma) \tag{4}$$

where $\mathbf{w}_{ij}$ is the new value for dimension i of the Brownian individual's centroid at index j, $\mathbf{y}_{ij}$ is the value for dimension i of the global best individual's centroid at index j and $N(0, \sigma)$ is Gaussian noise with mean 0 and standard deviation σ.

The CDCDynDE is summarised in Algorithm 1.

IV. Experimental Setup

Auto-generated temporal datasets provided by Graaff [6] were used. The datasets include three data migration types, namely pattern migration, cluster migration and centroid migration. For pattern migration a single data pattern migrates from the cluster to which it belongs to a randomly selected cluster. For cluster migration all data patterns in a selected cluster migrate to other randomly selected clusters. For centroid migration all data patterns of a cluster migrate to new positions where the patterns still belong to the same cluster. Clusters appear, disappear, grow and shrink in the various datasets.

The datasets consisted of 8000 patterns, 100 time-steps, 80 data patterns per time-step and a maximum of eight clusters at any point in time. Patterns consisted of either three, eight or fifteen attributes. Frequencies, which refer to the rate at which changes occur, ranged from one to five, where a larger number implies less changes. The severities of change, which refer to the magnitude of the change, ranged from one to five. The iteration at which a change occurs was calculated as $\frac{f}{10} * T$, where f is the frequency of change, and T is the total number of iterations. A total of 225 artificial datasets were clustered as there are three migration types, three dimensions, five frequencies and five severities.

Algorithm 1 CDCDynDE

Initialise S sub-populations of individuals
while stopping condition is not reached **do**
  Generate a context individual by replacing each dimension with the best solution of each population respectively, and denote as $b(k, S_k.\hat{\mathbf{y}}_i)$, where k is the population index and $S_k.\hat{\mathbf{y}}_i$ is the best solution for that population.
  for each sub-population S_k **do**
    for each individual $S_k.\mathbf{x}_i$ **do**
      Replace dimension k of $b(k, S_k.\hat{\mathbf{y}}_i)$ with $S_k.\mathbf{x}_i$ to create $b(k, S_k.\mathbf{x}_i)$
      for each data pattern $\mathbf{z}_p$ **do**
        Calculate the Euclidean $d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j})$ of $\mathbf{z}_p$ and each cluster centroid $S_k.\mathbf{x}_{ik_j}$ of $b(k, S_k.\mathbf{x}_i)$
        Assign pattern $\mathbf{z}_p$ to centroid $S_k.\mathbf{x}_{ik_j}$, such that $d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j}) = min_{\forall k_j = 1 \ldots N_{k_j}} \{d(\mathbf{z}_p, S_k.\mathbf{x}_{ik_j})\}$
      end for
      Calculate the fitness of $b(k, S_k.\mathbf{x}_i)$
    end for
  end for
  Compare the global best individuals from each sub-population to each other
  if the global best positions of both sub-populations are within r_{excl} of each other **then**
    re-initialise the sub-population with the worst global best
  else
    for each sub-population that was not re-initialised **do**
      for each individual $S_k.\mathbf{x}_i$ **do**
        Update $S_k.\mathbf{x}_i$ using the mutation, crossover and selection mechanisms of the DE
      end for
      Update context individual
      Make a percentage of the weakest individuals Brownian Individuals and update them using Equation (4)
    end for
  end if
end while

A population of 50 individuals was initialised within the bounds of the dataset, which are defined per dimension by

the lowest and highest value of each attribute in the dataset. Algorithms were run for a total of 1000 iterations and results were averaged over 30 independent runs. Centroids leaving the boundaries of the search space were re-set to remain on the boundary. Ten percent of the population was re-initialised in the RDCDE and 10% of the population was selected to become Brownian individuals in both the DCDynDE and the CDCDynDE. An exclusion radius of 5.0 was used along with a scaling factor and crossover probability of 0.5. The DCDynDE and CDCDynDE both used a total of eight populations as the maximum number of clusters in the datasets at any point in time was eight.

The quality of the clustering solution was measured using three measures [6], namely the inter-cluster distance, the intra-cluster distance and the Ray-turi validity index. The inter-cluster distance refers to the distance between clusters, which is maximised and calculated using

$$J_{iter} = \frac{2}{n_K(n_K-1)} \sum_{k=1}^{n_K-1} \sum_{k_2=k+1}^{n_K} d(\mathbf{c}_k, \mathbf{c}_{k_2}) \quad (5)$$

where $\mathbf{c}_k$ and $\mathbf{c}_{k_2}$ are cluster centroids.

The intra-cluster distance refers to the distance between data patterns within a cluster. This distance is minimised and calculated using

$$J_{intra} = \frac{\sum_{k=1}^{n_K} \sum_{\forall \mathbf{z} \in C_k} d(\mathbf{z}_p, \mathbf{c}_k)}{|P|} \quad (6)$$

where $|P|$ is the total number of data patterns in the dataset.

The last measure is the Ray-turi validity index. This measure combines the inter-cluster and intra-cluster distances for a more accurate representation of the quality of the clustering solution. The Ray-turi validity index is minimised and calculated using

$$Q_D = \frac{J_{intra}}{inter_{min}} \quad (7)$$

where $inter_{min}$ is the smallest inter-cluster distance found using Equation (5).

Wins and losses per algorithm based on the Ray-turi validity index were determined based on an approach proposed by Helbig and Engelbrecht [26]. For each sample, the Ray-turi validity index of all iterations before a change to the environment occurred was averaged. This gives a more accurate representation of the performance of the algorithm over time than only taking the last iteration's value. For each problem each algorithm's performance was compared against each other algorithm using 30 independent samples. First a Kruskal-Wallis test was performed to determine if there is a statistically significant difference between the algorithms' performance. If there was a statistically significant difference, pair-wise Mann-Whitney U tests were performed and the resulting U-values were used to determine the winning and losing algorithm for a particular problem.

V. Results and Discussion

The average inter-cluster distance, intra-cluster distance and Ray-turi validity index values are reported in Table I. The table shows that the CDCDynDE obtained the highest inter-cluster distance and, therefore, had the most separated clusters. This result has a large standard deviation value, implying that the resulting inter-cluster distances varied significantly for the problems used. The DCDynDE, on the other hand, obtained the lowest inter-cluster distance making its resulting clusters the least separate. Lastly, the SDCDE and RDCDE obtained similar values for the inter-cluster distance.

The DCDynDE obtained the lowest intra-cluster distance and, therefore, the most compact clusters. The CDCDynDE, SDCDE and RDCDE clusters resulted in similar values for the intra-cluster distance, with the RDCDE obtaining the highest value. Once again, CDCDynDE has a large standard deviation, implying a significant difference between the resulting intra-cluster distances for the problems used.

The Ray-turi validity measures the performance of the algorithm by combining the inter-cluster and intra cluster distance measures. This is a more accurate measure because a clustering solution is required to have clusters that are very compact and highly separated from each other. The CDCDynDE obtained the lowest Ray-turi validity index, indicating that the algorithm performed the clustering task most effectively. The variation of the Ray-turi validity index across all the problems is much less significant than that of the inter-cluster and intra-cluster distances for the CDCDynDE algorithm. The DCDynDE performed the clustering task better than the SDCDE and RDCDE, but worse than the CDCDynDE.

Table II summarises the total statistically significant wins, $\#wins$, and losses, $\#losses$, obtained by each algorithm as well as the $Diff$ value, which is calculated using $Diff = \#wins - \#losses$. The CDCDynDE obtained the most wins and least losses when compared against the other algorithms. The DCDynDE obtained 70 more losses than wins, while the SDCDE and RDCDE both obtained significantly more losses than wins. These results are illustrated in Figure 1 for a visual interpretation.

The problems used consisted of various severities, frequencies, dimensions and pattern migration types. Figures 2-5 illustrate the behaviour of each algorithm as the various parameters change.

Figure 2 illustrates $Diff$ of each algorithm for dimensions 3, 8 and 15. For the three dimensional problems the CDCDynDE and the DCDynDE both have a high $Diff$ and, therefore, significantly more wins than losses. The SDCDE and RDCDE, on the other hand, obtained significantly more losses than wins for three-dimensional problems. As the dimensions increase the performance of the DCDynDE abruptly decreases, while the performance of the SDCDE and RDCDE slightly increases. The CDCDynDE remains the best performing algorithm throughout, obtaining the most wins and least losses for all-dimensional problems, and is the only algorithm with a positive $Diff$ value throughout.

TABLE I

AVERAGES AND STANDARD DEVIATION FOR INTER-CLUSTER DISTANCE, INTRA-CLUSTER DISTANCE AND RAY-TURI VALIDITY FOR EACH ALGORITHM

Algorithm	**Inter-cluster Distance**	**Intra-cluster Distance**	**Ray-Turi Validity**
SDCDE	22.983 $\pm$ 3.266	10.679 $\pm$ 3.981	1.141 $\pm$ 0.178
RDCDE	22.963 $\pm$ 3.257	10.682 $\pm$ 3.979	1.154 $\pm$ 0.198
DCDynDE	19.830 $\pm$ 3.346	**3.877** $\pm$ **2.995**	0.979 $\pm$ 0.207
CDCDynDE	**30.072** $\pm$ **11.323**	10.39107 $\pm$ 8.908	**0.677** $\pm$ **0.276**

TABLE II

TOTAL STATISTICALLY SIGNIFICANT WINS AND LOSSES PER ALGORITHM

Attribute	**SDCDE**	**RDCDE**	**DCDynDE**	**CDCDynDE**
Wins	203	204	300	632
Losses	466	466	370	37
Wins - losses	-263	-262	-70	595

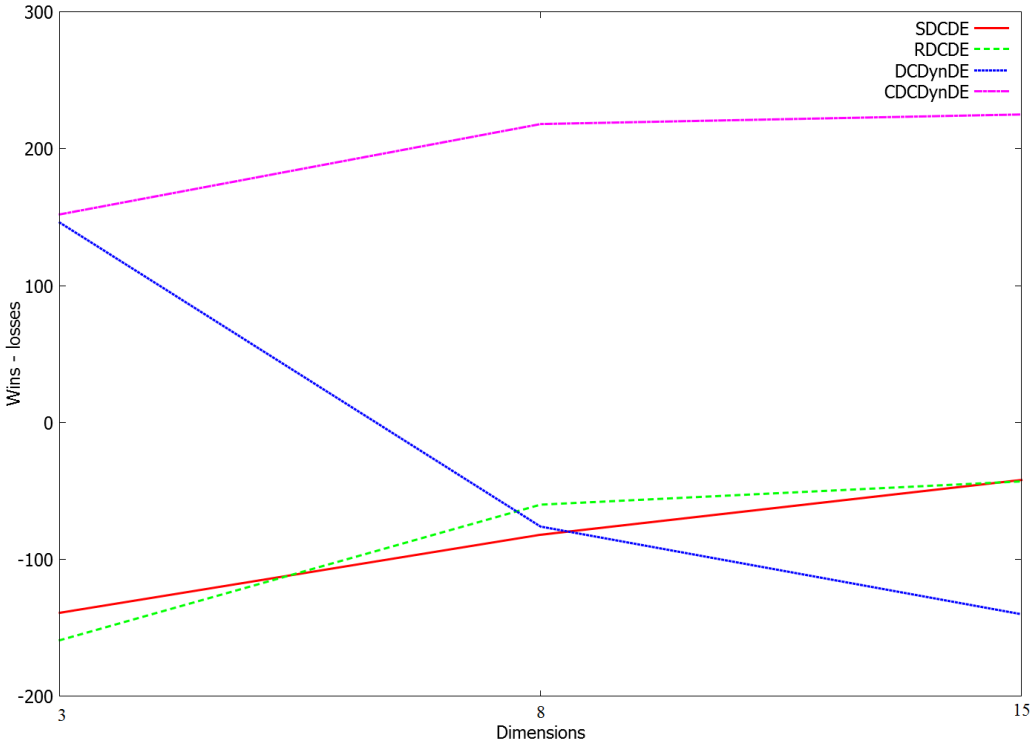


Fig. 2. $Diff$ per algorithm as dimensions change

Figure 3 illustrates $Diff$ of each algorithm for frequencies 1, 2, 3, 4 and 5, where a higher value means that less changes occur in the environment. The CDCDynDE obtained the highest $Diff$ value for all frequencies. The DCDynDE, on the other hand, obtained more losses than wins when less changes occurred in the environment and improved performance as the changes increased (with lower frequency values). The SDCDE and RDCDE both obtained more losses than wins for all frequencies, decreasing performance as more changes in the environment occurred.

Figure 4 illustrates $Diff$ of each algorithm for severities 1, 2, 3, 4 and 5. The CDCDynDE obtained the most wins and the least losses for all severities of change. The DCDynDE obtained increased performance on higher severities, but still remained with more losses than wins for the majority of the severities, while the CDCDynDE's performance decreased by a small amount.

Figure 5 illustrates the total wins and losses per algorithm for pattern migration, centroid migration and cluster migration problems. The CDCDynDE obtained the most wins and least losses over all three migration types. The DCDynDE, SDCDE and RDCDE all obtained more losses than wins for

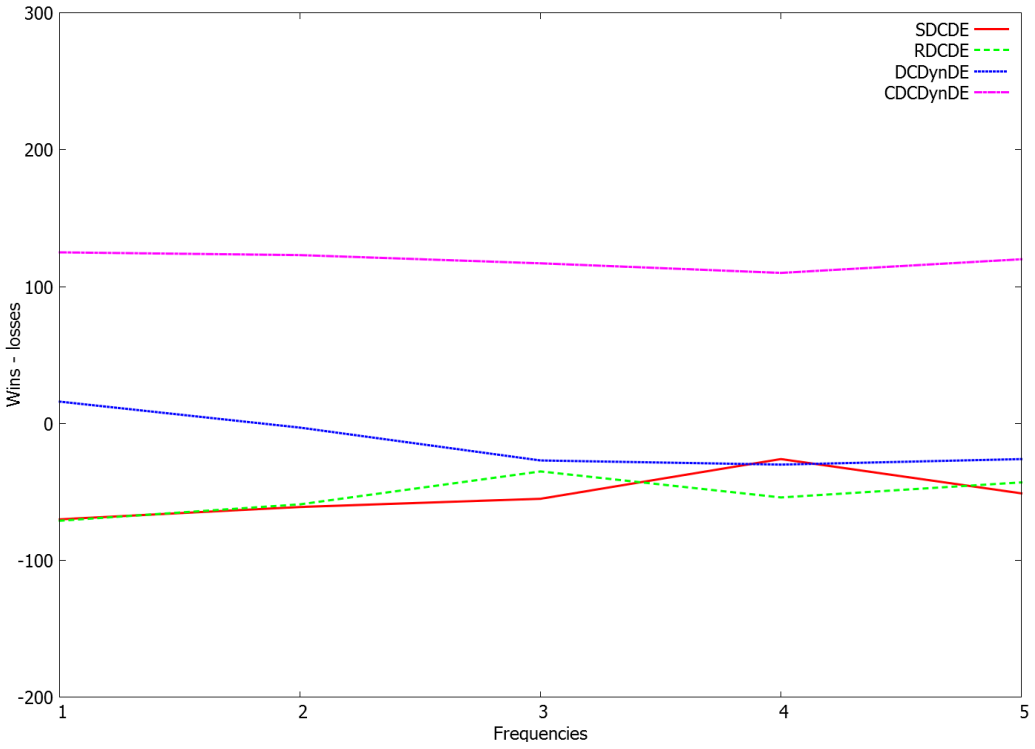


Fig. 3. $Diff$ per algorithm as frequencies change

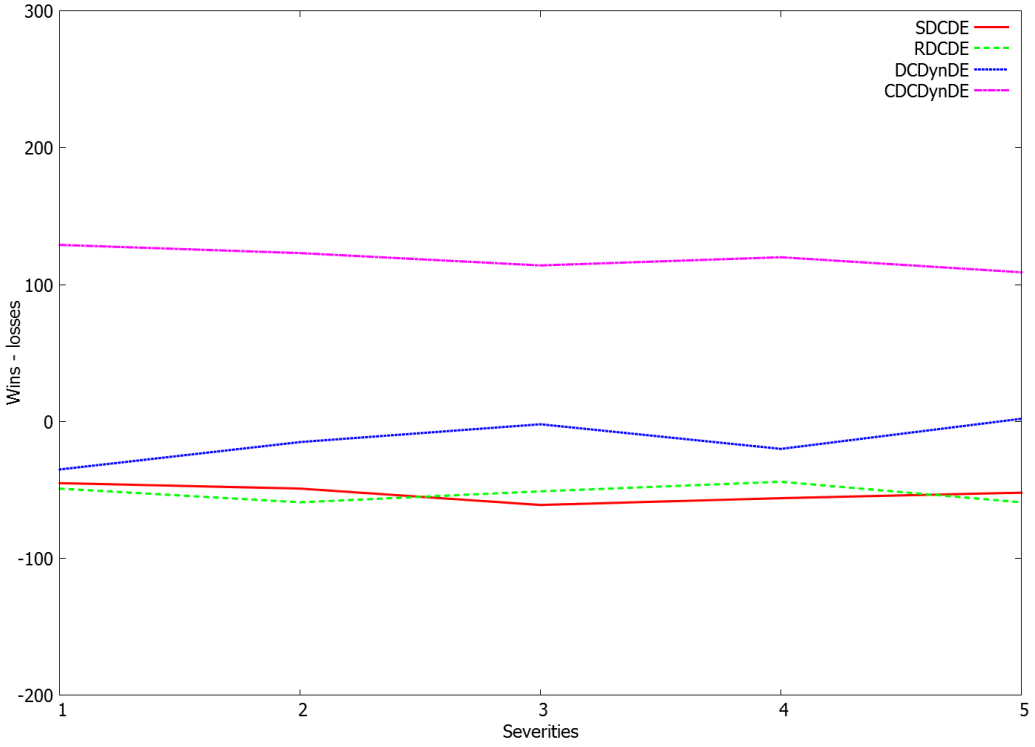


Fig. 4. $Diff$ per algorithm as severities change

pattern migration problems. The DCDynDE improved performance compared to the SDCDE and RDCDE when used on centroid migration and cluster migration problems, but still obtained more losses than wins due to the CDCDynDE

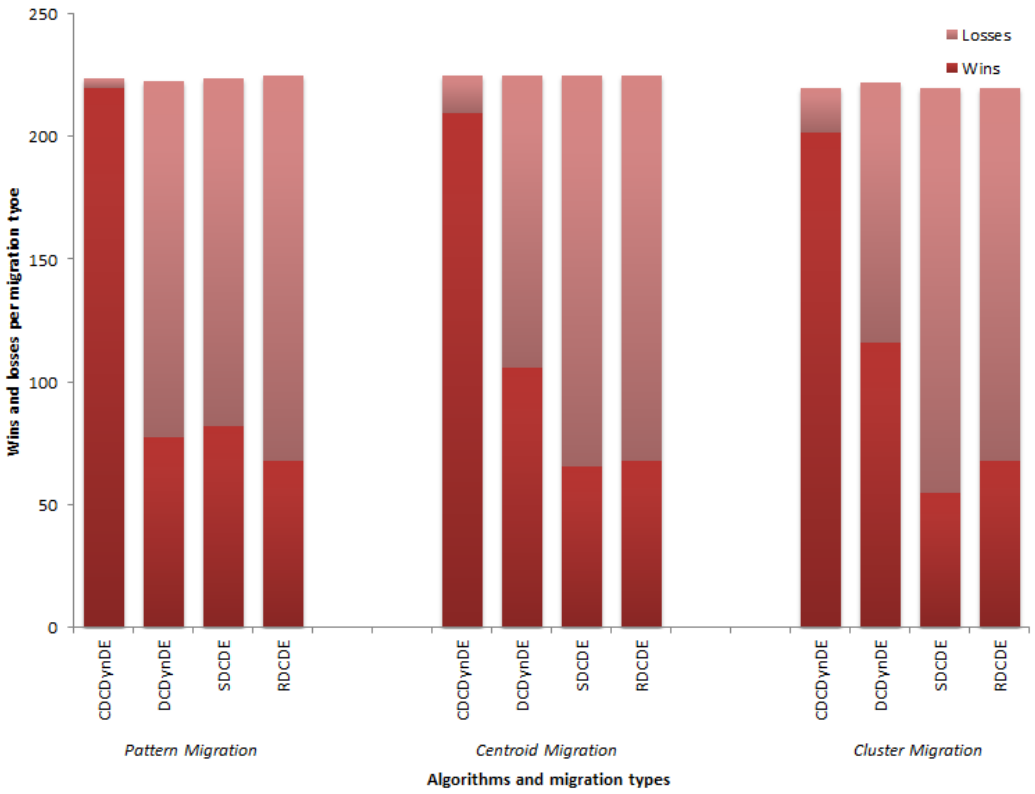


Fig. 5. Wins and losses per algorithm for different migration types

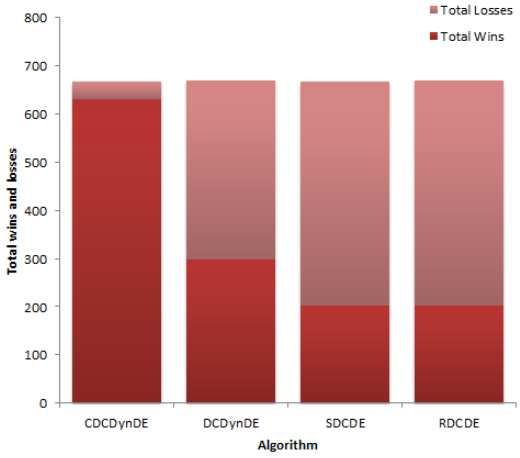


Fig. 1. Total wins and losses per algorithm over all problems

performing better on all three migration types.

VI. Conclusions and Future Work

This article introduced the CDCDynDE, an adaptation to Georgieva and Engelbrecht's [13] DCDynDE. It evaluated the performance of the proposed algorithm against the DCDynDE, SDCDE and RDCDE applied to a variety of temporal datasets. The datasets included different severities of change, frequencies of change, dimensions and pattern migration types.

The average inter-cluster distance showed that the CDCDynDE obtained the most separable clusters, while the average intra-cluster distance showed that the DCDynDE obtained the most compact clusters. However, both measures need to be considered in order to determine the quality of a clustering solution. For this reason the Ray-turi validity index was used to analyse the quality of the clustering solutions in more detail. The average Ray-turi validity index value showed that the CDCDynDE performed the clustering task better than the DCDynDE, SDCDE and RDCDE.

Statistical tests were performed to determine the total wins and losses obtained by each algorithm when compared to each other algorithm. The Ray-turi validity index was used for these tests. The CDCDynDE obtained the most wins and least losses overall. When analysed in more detail it was shown that the CDCDynDE obtained the most wins and losses for all changes in severity, dimension, frequency and migration type.

Future work may include comparing the algorithm to other temporal data clustering algorithms, such as PSO, SOM and AIS. A more detailed study can be performed to analyse the algorithm's performance over various parameters such as frequency, severity and dimension. The CDCDynDE can also be adapted to allow it to optimize the number of clusters and can be applied to real-world datasets. Lastly, an analysis of the effect of regenerating the context individual when the best individual of the population is updated, instead of at the

beginning of every iteration, can be done.

ACKNOWLEDGEMENT

The support of SAP P&I BIT Mobile Empowerment and the SARChI research chair in AI is hereby acknowledged. The views and conclusions expressed in this paper are those of the authors and are not necessarily to be attributed to the mentioned parties.

REFERENCES

[1] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, 2009.

[2] J. Herrero, A. Valencia, and J. Dopazo, "A hierarchical unsupervised growing neural network for clustering gene expression patterns.," *Bioinformatics*, vol. 17, no. 2, pp. 126–136, 2001.

[3] D. W. Van Der Merwe and A. P. Engelbrecht, "Data clustering using particle swarm optimization," in *2003 Congress on Evolutionary Computation 2003 CEC 03 (2003)*, vol. 1, pp. 215–220, 2003.

[4] B. Santosa and M. K. Ningrum, "Cat swarm optimization for clustering," in *Soft Computing and Pattern Recognition, 2009. SOCPAR'09. International Conference of*, pp. 54–59, IEEE, 2009.

[5] D. Karaboga and C. Ozturk, "A novel clustering approach: Artificial bee colony (abc) algorithm," *Applied Soft Computing*, vol. 11, no. 1, pp. 652–657, 2011.

[6] A. Graaff, *A Local Network Neighbourhood Artificial Immune System*. PhD thesis, University of Preotria, June 2011.

[7] L. N. de Castro and F. J. Von Zuben, "ainet: an artificial immune network for data analysis," *Data mining: a heuristic approach*, vol. 1, pp. 231–259, 2001.

[8] P. Scheunders, "A genetic c-means clustering algorithm applied to color image quantization," *Pattern Recognition*, vol. 30, no. 6, pp. 859–866.

[9] S. A. Hanuman, V. A. Babu, A. Govardhan, and S. C. Satapathy, "Data clustering using almost parameter free differential evolution technique," *International Journal of Computer Applications*, vol. 8, pp. 1–7, October 2010.

[10] J. Patel, "Temporal database system," Master's thesis, Department of Computing, Imperial College, University of London,, June 2003.

[11] T. Blackwell, "Particle swarm optimization in dynamic environments," *Evolutionary Computation in Dynamic and Uncertain Environments*, vol. 49, pp. 29–49, 2007.

[12] K. Georgieva and A. P. Engelbrecht, "A cooperative multi-population approach to clustering temporal data," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1983–1991, 2013.

[13] K. Georgieva and A. P. Engelbrecht, "Dynamic differential evolution algorithm for clustering temporal data," in *9th International Conference on Large-Scale Scientific Computations*, 2013.

[14] M. Van Der Voort, M. Dougherty, and S. Watson, "Combining kohonen maps with arima time series models to forecast traffic flow," *Transportation Research Part C: Emerging Technologies*, vol. 4, no. 5, pp. 307–318, 1996.

[15] R. Storn, "On the usage of differential evolution for function optimization," in *Fuzzy Information Processing Society, 1996. NAFIPS. 1996 Biennial Conference of the North American*, pp. 519–523, IEEE, 1996.

[16] M. G. H. Omran, A. P. Engelbrecht, and A. Salman, "Differential evolution methods for unsupervised image classification," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 2, pp. 966–973, IEEE, 2005.

[17] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed-forward neural networks," *Neural Processing Letters*, vol. 17, no. 1, pp. 93–105, 2003.

[18] R. Storn and K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[19] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *Evolutionary Computation, IEEE Transactions on*, vol. 15, no. 1, pp. 4–31, 2011.

[20] H.-Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *Journal of Global Optimization*, vol. 27, no. 1, pp. 105–129, 2003.

[21] F. Neri and V. Tirronen, "Recent advances in differential evolution: a survey and experimental analysis," *Artificial Intelligence Review*, vol. 33, no. 1-2, pp. 61–106, 2010.

[22] D. W. van der Merwe and A. P. Engelbrecht, "Data clustering using particle swarm optimization," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 1, pp. 215–220, 2003.

[23] M. C. du Plessis and A. P. Engelbrecht, "Improved differential evolution for dynamic optimization problems," in *IEEE Congress on Evolutionary Computation IEEE World Congress on Computational Intelligence*, 2008.

[24] R. Mendes and A. S. Mohais, "Dynde: a differential evolution for dynamic optimization problems," in *IEEE Congress on Evolutionary Computation*, vol. 3, pp. 2808–2815, 2005.

[25] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," in *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 225–239, 2004.

[26] M. Helbig and A. P. Engelbrecht, "Issues with performance measures for dynamic multi-objective optimisation," in *2013 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*, pp. 17–24, IEEE, 2013.