

UNIVERSIDADE FEDERAL DO PARANÁ

ADRIANO FRANCISCO KULTZAK

AN EXPLORATORY STUDY ON THE BEHAVIOR OF DIFFERENT MULTIOBJECTIVE
ALGORITHMS IN CLUSTERING

CURITIBA PR

2020

ADRIANO FRANCISCO KULTZAK

AN EXPLORATORY STUDY ON THE BEHAVIOR OF DIFFERENT MULTIOBJECTIVE
ALGORITHMS IN CLUSTERING

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Aurora Trinidad Ramirez Pozo.

Coorientador: Marcilio Carlos Pereira de Souto.

CURITIBA PR

2020

CATALOGAÇÃO NA FONTE – SIBI/UFPR

P96

Kultzak, Adriano Francisco

An exploratory study on the behavior of different multiobjective algorithms in clustering [recurso eletrônico]/ Adriano Francisco Kultzak. Curitiba, 2020.

Dissertation (Master) - Master of Sciences in Informatics in the Graduate Program in Informatics, Exact Sciences Sector, of the Federal University of Paraná, Brazil

Advisor: Aurora Trinidad Ramirez Pozo.

Co-advisor: Marcilio Carlos Pereira de Souto.

1. Computer Algorithms. 2. Computer Science. I. Pozo, Aurora Trinidad Ramirez. II. Souto, Marcilio Carlos Pereira de. I. Título.

CDD 005.1

Bibliotecária: Vilma Machado CRB9/1563

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **ADRIANO FRANCISCO KULTZAK** intitulada: **AN EXPLORATORY STUDY ON THE BEHAVIOR OF DIFFERENT MULTIOBJECTIVE ALGORITHMS IN CLUSTERING**, sob orientação da Profa. Dra. AURORA TRINIDAD RAMIREZ POZO, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua aprovação no rito de defesa. A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 10 de Março de 2020.



AURORA TRINIDAD RAMIREZ POZO

Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)



KATTI FACELI

Avaliador Externo (UNIVERSIDADE FEDERAL DE SÃO CARLOS)



MARCILIO CARLOS PEREIRA DE SOUTO

Coordenador - Avaliador Externo (UNIVERSIDADE DE ORLÉANS - FRANÇA)



CARMEM SATIE HARA

Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)



To everyone I could learn from.

ACKNOWLEDGEMENTS

First I would like to thank my family for showing me the value of education and given me the strength to accomplish this work. I am deeply grateful to my advisors Aurora Pozo e Marcílio de Souto for counting on me as part of this research, and for all the learning I have had with them. I extend my thanks to the DInf teachers who contributed to my formation and the members of the examining board for their valuable contributions. I would also like to thank the friends of the Bioinspired Computing group (C-BIO) and the Software Engineering group (GRES) for their support.

RESUMO

Algoritmos de clustering tradicionais tem dificuldades para encontrar padrões em conjuntos de dados que apresentam estruturas heterogêneas pois cada algoritmo de acordo com a heurística implementada possui um viés para identificação de determinados formatos de grupo. Buscando solucionar esse problema surge a técnica do ensemble de agrupamentos que consiste em combinar um conjunto diverso de partições produzidas por algoritmos baseados em diferentes critérios de clustering e obter uma partição consenso que melhor represente as diversas estruturas presentes nos dados. Outra abordagem para superar as limitações dos algoritmos tradicionais é o clustering multi-objetivo, que otimiza simultaneamente mais de uma função objetivo, aproveitando-se também dos diferentes critérios de clustering que essas funções expressam. Este estudo apresenta a abordagem jMocle, contruída sobre o framework multi-objetivo jMetal e sobre a técnica que alia clustering multi-objetivo e cluster ensemble, MOCLE. Os experimentos realizados com o jMocle comparam diferentes algoritmos multi-objetivo no contexto de clustering e avaliam a qualidade das soluções geradas considerando a evolução das soluções. Os experimentos avaliam também a qualidade das partições geradas de acordo com o índice Adjusted Rand e comparam resultados com os algoritmos estado-da-arte em clustering multi-objetivo MOCK e Δ -MOCK. Os resultados não evidenciam diferença estatística significativa, portanto não é possível afirmar haver um algoritmo multi-objetivo exclusivamente melhor para clustering. Tendo em vista a não identificação de evidência estatística de diferença entre os algoritmos multi-objetivo, então foi realizada uma análise qualitativa dos métodos MOCLE, MOCK e Δ -MOCK, considerando o desempenho dos algoritmos de acordo com os diferentes tipos de conjunto de dados utilizados. Nesta análise são exploradas características dos algoritmos como o impacto do procedimento de inicialização, cruzamento e mutação e a relevância do parâmetro do número de vizinhos na identificação de determinados tipos de estrutura nos dados.

Palavras-chave: Clustering. Ensemble de agrupamentos. Otimização multi-objetivo.

ABSTRACT

Traditional clustering algorithms have difficulties in finding patterns in datasets that have heterogeneous structures because every algorithm according to the implemented heuristic has a bias towards identifying particular cluster shapes. To solve this problem, the cluster ensemble technique consists of combining a diverse set of partitions produced by algorithms based on different clustering criteria and obtaining a consensus partition that best represents the various structures present in the data. Another approach that seeks to overcome the limitations of traditional algorithms is multi-objective clustering, which simultaneously optimizes more than one objective function, taking advantage of the different clustering criteria that these functions express. This study presents the jMocle approach, built over the jMetal multi-objective framework and over the technique that combines multi-objective clustering and cluster ensemble, MOCLE. The experiments performed with jMocle compare different multi-objective algorithms in the context of clustering and evaluate the quality of the solutions generated considering their evolution. The experiments also evaluate the quality of the generated partitions according to the Adjusted Rand index and compare results with the multi-objective state-of-the-art clustering algorithms MOCK and Δ -MOCK. The results do not show statistically significant differences, therefore it is not possible to say that there is a multi-objective algorithm exclusively better for clustering. Given the non-identification of evidence of statistical difference between the multi-objective algorithms, a qualitative analysis of the MOCLE, MOCK and Δ -MOCK methods was performed, considering the performance of the algorithms according to the different types of data sets used. In this analysis, characteristics of the algorithms are explored such as the impact of the initialization, crossover and mutation procedure and the relevance of the number of neighbors parameter in identifying certain types of structure in the data.

Keywords: Clustering. Cluster ensemble. Multi-objective optimization.

LIST OF FIGURES

2.1	Illustration of k -means clustering.	17
2.2	Example of dendrogram.	18
2.3	Illustration of Ward-linkage clustering	19
2.4	Example of SNN density classification. Adapted from [24]	20
2.5	Example of condensed dendrogram. [64]	21
2.6	Dataset with multiple structures. Adapted from [26].	22
2.7	Illustration of MCLA. [68]	23
2.8	Illustration of HBGF. [32]	24
2.9	CSPA similarity matrix construction. Adapted from [68]	25
3.1	Illustration of Pareto dominance. Adapted from [61]	29
3.2	Illustration of a Pareto non-dominated front. Adapted from [61]	29
3.3	Illustration of the MOEA evolution of solutions in a minimization problem.	31
3.4	Classification of multi-objective evolutionary algorithms. Adapted from [55]	31
3.5	Illustration of PBI decomposition.	33
3.6	Illustration of Tchebycheff decomposition	33
3.7	Crowding comparison. Adapted from [20]	35
3.8	Reference Point Structure. Adapted from [18]	36
3.9	Illustration of SPEA2+SDE density estimation. Adapted from [59]	39
3.10	Example of Stable marriage problem. Adapted from [58]	43
3.11	Two layer reference point structure. Adapted from (79)	47
3.12	Illustration of hypervolume covering area. Adapted from [85].	51
4.1	MST-based Representation. Adapted from [39]	54
4.2	Δ -MOCK Representations. Adapted from [36]	55
5.1	Overall structure of jMocle used to evaluate different MOEAs	59
5.2	Solution encoding scheme.	61
6.1	ds2c2sc13 dataset structures.	63
6.2	Critical Difference in the hypervolume using HBGF crossover.	68
6.3	Critical Difference in the Adjusted Rand Index.	69
6.4	Critical Difference in the hypervolume using MCLA crossover.	70
6.5	Critical Difference in the ARI using MCLA crossover.	71
6.6	Initial population vs final population with HBGF crossover - ds2c2sc13_E3.	73

6.7	Initial population vs final population with HBGF crossover - D31.	73
6.8	Initial population vs final population with HBGF crossover - Frogs_MFCC_E3.	74
6.9	Initial population vs final population with HBGF crossover - Golub_E3.	74
6.10	Initial population vs final population with MCLA crossover - ds2c2sc13_E3.. . . .	75
6.11	Initial population vs final population with MCLA crossover - OptDigits.	75
6.12	Random initial population - ds3c3sc6_E2.	76
6.13	Random initial population vs jMocle initial population - ds3c3sc6_E2.	77
6.14	Comparison between MOCLE and MOCK/MOCK-D - ds2c2sc13_E3.	79
6.15	Comparison between MOCLE and MOCK/MOCK-D - OptDigits.	80
6.16	Comparison between MOCLE and MOCK/MOCK-D - Golub_E3.	80
7.1	D31: median best partition found by MOCLE ($ARI = 0.953$ and $k = 31$). Points with the same marker (1,...,31) belong to the same cluster.	86
7.2	D31: median best partition found by Δ -MOCK ($ARI = 0.757$ and $k = 29$). Points with the same marker (1,...,29) belong to the same cluster..	86
7.3	D31: median best partition found by MOCK ($ARI = 0.860$ and $k = 41$). Points with the same marker (1,...,41) belong to the same cluster.	87

LIST OF TABLES

2.1	Summary comparison of clustering algorithms.	27
3.1	Classification of studied MOEAs	52
5.1	Example data-set	60
6.1	Artificial Dataset Description. Number of objects n , number of attributes d , and number of clusters k^*	62
6.2	Real Dataset Description. Number of objects n , number of attributes d , and number of clusters k^*	63
6.3	Average Rankings and standard deviation obtained in the Friedman test of MOEAs hypervolume results across the 19 datasets using HBGF crossover	67
6.4	Average Rankings and standard deviation obtained in the Friedman test of MOEAs ARI results across the 19 datasets using HBGF crossover	68
6.5	Average Rankings and standard deviation obtained in the Friedman test of MOEAs hypervolume results across the 12 datasets using MCLA crossover	69
6.6	Average Rankings and standard deviation obtained in the Friedman test of MOEAs ARI results across the 12 datasets using MCLA crossover	70
6.7	Mean and standard deviation for the number of solutions produced in 30 jMocle+HBGF runs. In bold are the initial population size	71
6.8	Identification of different data structures in a single dataset using HBGF crossover. ARI, number of clusters between parenthesis.	72
6.9	Best ARI score in 30 executions for each problem	77
6.10	Best ARI score obtained from 30 jMocle runs in artificial datasets	78
6.11	Best ARI score obtained from 30 jMocle runs in real datasets	78
7.1	Dataset characteristics: number of objects n , number of attributes d , number of specified structures n^E , and number of clusters K for each structure E_i	82
7.2	ARI (number of clusters) of the best partition found by MOCLE, MOCK and Δ -MOCK. Average of 30 executions.	85
7.3	ARI (number of clusters) of the best partition found by k -means, the Ward linkage method, SNN and HDBSCAN.	87

LIST OF ACRONYMS

ARI	Adjusted Rand Index
ASF	Achievement Scalarizing Function
BCE	Bi-criterion Evolution
CSPA	Cluster-based Similarity Partitioning Algorithm
DBSCAN	Density Based Spatial Clustering of Applications with Noise
GrEA	Grid-based Evolutionary Algorithm
HBGF	Hybrid Bipartite Graph Formulation
HEA-DP	Hybrid Evolutionary Algorithm with Dual Populations
HypE	Hypervolume Estimation algorithm for multi-objective optimization
IBEA	Indicator Based Evolutionary Algorithm
jMocle	jMetal Based Multiobjective Cluster Ensemble
KnEA	Knee Point Driven Evolutionary Algorithm
LIBRAS	Linguagem Brasileira de Sinais
MCLA	Meta-clustering Algorithm
MOCK	Multiobjective Clustering with automatic K-determination
MOCLE	Multiobjective Cluster Ensemble
MOEA	Multiobjective Evolutionary Algorithm
MOEA\D	Multiobjective Evolutionary Algorithm Based on Decomposition
MOEA\DD	Multiobjective Evolutionary Algorithm Based on Dominance and Decomposition
MOMBI	Many-Objective Metaheuristic Based on the R2 Indicator
MST	Minimum Spanning Tree
NSGA	Non-dominated Sorting Genetic Algorithm
PBI	penalty Based Boundary Intersection
PCA	Principal Component Analysis
PESA	Pareto Envelope-based Selection Algorithm
SBX	Simulated Binary Cross-over
SDE	Shift-based Density Estimation
SNN	Shared Nearest Neighbor
SPEA	Strength Pareto Evolutionary Algorithm
SSE	Sum of Squared Errors
STM	stable matching model
θ -DEA	θ Dominance-based Evolutionary Algorithm

LIST OF SYMBOLS

c	Cluster
d	Number of dimensions
$d(x, y)$	Distance between x and y
G	Graph
H	Number of division in a objective axis
g^{pbi}	PBI function
g^{te}	Tchebychef function
k^*	Number of clusters in the true partition
K	Number of clusters
K_{min}	Minimum number of clusters
K_{max}	Maximum number of clusters
L	Number of nearest neighbors
m	Number of objective functions
N	Number of objects in a dataset
nn_{ij}	j -th nearest neighbour of object $\mathbf{x}_i$
odv	Overall deviation function
R^m	Objective Space
V	Vertex
var	Intra-cluster Variance function
x	Object in dataset
z^*	Ideal Objective Vector
z^{nad}	Nadir Vector
α	Attainment function
λ	Weight vector
π	Partition
π^F	Consensus Partition
μ	Cluster centroid
Ω	Decision Space
σ_i^k	Distance between the k -th element and i
Δ	Set of non-relevant fixed links
Γ	Set of relevant links

CONTENTS

1	INTRODUCTION	14
1.1	MOTIVATION	14
1.2	OBJECTIVES.	15
1.3	WORK ORGANIZATION	15
2	CLUSTERING.	16
2.1	CLUSTERING DEFINITION	16
2.1.1	Partitional Clustering	16
2.1.2	Hierarchical Clustering	18
2.1.3	Density-based Clustering	19
2.2	CLUSTER ENSEMBLE	21
2.2.1	MCLA.	23
2.2.2	HBGF	24
2.2.3	CSPA	24
2.3	CLUSTER VALIDATION	25
2.3.1	Adjusted Rand Index	25
2.4	DISCUSSION.	26
3	MULTI-OBJECTIVE OPTIMIZATION	28
3.1	MULTI-OBJECTIVE OPTIMIZATION	28
3.1.1	Pareto Dominance.	28
3.2	MULTI-OBJECTIVE EVOLUTIONARY ALGORITHMS	30
3.2.1	NSGA-II.	34
3.2.2	NSGA-III	35
3.2.3	SPEA2	37
3.2.4	SPEA2+SDE	38
3.2.5	MOEA/D	40
3.2.6	MOEA/DD	41
3.2.7	MOEA/D-STM	43
3.2.8	MOMBI-II	44
3.2.9	θ -DEA.	46
3.2.10	IBEA	48
3.2.11	HypE	49
3.3	HYPERVOLUME INDICATOR	51
3.4	DISCUSSION.	52

4	RELATED WORKS.	53
4.1	MOCK	53
4.2	Δ -MOCK	55
4.3	MOCLE	56
4.4	DISCUSSION.	57
5	PROPOSED APPROACH.	59
5.1	APPROACH DEFINITION.	59
5.2	DISCUSSION.	61
6	EVALUATION OF JMOCLE.	62
6.1	DATASETS	62
6.2	GENERATION OF THE BASE PARTITION SET	64
6.3	MULTIOBJECTIVE CLUSTERING ALGORITHMS PARAMETERS.	65
6.3.1	jMocle.	65
6.3.2	MOCK and Δ -MOCK.	66
6.4	PERFORMANCE ASSESSMENT.	66
6.5	RESULTS AND ANALYSIS	67
6.6	DISCUSSION.	81
7	DATA DRIVEN ANALYSIS	82
7.1	EXPERIMENTAL DESIGN	82
7.1.1	Datasets	82
7.1.2	Performance assessment	83
7.1.3	Experimental Setup	83
7.2	RESULTS AND DISCUSSION	84
7.2.1	Clustering performance	84
7.2.2	Number/size of clusters of the best partitions	88
7.3	DISCUSSION.	89
8	CONCLUSION AND FUTURE WORK.	90
8.1	FUTURE WORK	90
	REFERENCES	92

1 INTRODUCTION

Clustering is a type of unsupervised learning technique, in other words, it is a technique that does not require prior knowledge of the classes or categories to which the data belong. The goal of clustering is to find the underlying structure composed of clusters present in the data [49, 76, 1]. As there are no prior classes, a cluster can be defined as a set of objects presenting similarities within the same cluster and differences in patterns identified in different clusters [48].

An obvious way of clustering is, starting from the cluster definition, trying to identify groups of objects that are closer to each other using a distance measure, e.g., Euclidean distance, that is the measure used by k -means [63]. The heuristic implemented by k -means reflects a clustering criterion, in this case, compactness among components of a cluster. In addition to this criterion, there are others such as cluster density, separation, or compactness and separation simultaneously [65].

The traditional clustering algorithms have a bias for certain cluster shapes depending on the implemented heuristic and optimize only one clustering criterion (e.g., compactness of the clusters) being often very effective to this purpose. However, in general, they fail for data in accordance with a different criterion (e.g., chain-like and concentric clusters in the case of k -means). While an algorithm is best suited for hyper-spherical shapes, others tend to identify arbitrary shapes with similar density. The concept of cluster ensemble then arises. The technique of cluster ensemble consists in generate a diverse set of base partitions and using a consensus function to combine them. The use of cluster ensembles serves to obtain more robust clusters, which cannot be obtained by traditional algorithms, and less susceptible to the variability of each technique [68].

Another way to overcome traditional clustering algorithms is to optimize more than one objective function simultaneously to have a trade-off among those functions. The so-called multi-objective clustering results not in a single partition but a set of partitions [39].

While ensembles seek a single structure, they have a necessity for parameter setting and are susceptible to low-quality initial partitions. On the other hand, multi-objective approaches achieve multiple alternative outcomes and do not require many parameters adjustments. However, the greater the number of alternative outcomes, the more difficult it will be for a decision-maker to analyze them [26].

Among the various approaches to clustering, the technique Multi-objective cluster ensemble (MOCLE) aggregates the advantages of both multi-objective optimization and cluster ensemble [26].

1.1 MOTIVATION

Virtually most clustering algorithms assume a homogeneous criterion over the feature space, resulting in the detection of similar clusters among themselves, in terms of shape and density of each one [54]. Thus, the combination of multiple methods of clustering can take advantage of different objective functions and possibly bring better results.

The algorithm multi-objective clustering with automatic k -determination (MOCK) adopts the strategy of optimizing simultaneously two complementary objectives to form a set of solutions that is an approximation of the Pareto front [39].

Both MOCK and MOCLE have the optimization guided by a MOEA (multi-objective evolutionary algorithm), respectively PESA-II and NSGA-II. Although optimizing identical

objectives, Faceli's work, MOCLE, has a different initialization routine made by traditional clustering algorithms, another difference is in the crossover step that in MOCLE is performed by a cluster ensemble technique. These steps will be better addressed in the course of this work.

Efforts have been made to overcome traditional clustering algorithms, examples as MOCLE, MOCK and Δ -MOCK explore alternatives for improvement. However, the impact of many components of these algorithms is still unexplored, e.g., the influence of the optimization strategies, the objective functions, and other parameters.

1.2 OBJECTIVES

The objective of this work is to evaluate the behavior of MOCLE, technique that applies aspects of cluster ensemble and multi-objective clustering, having its optimization component, the Pareto-based algorithm NSGA-II, replaced by algorithms based on different approaches such as the relaxed dominance-based θ -DEA, reference-set-based NSGA-III, aggregation-based MOEA/D among others, as well as perform a data-driven analysis comparing the behavior of MOCLE, MOCK and Δ -MOCK.

The specific objectives of this work are:

- Implementation of a way of choosing between different MOEAs in the MOCLE.
- Evaluation, according to the quality of the fronts and quality of the partitions, the set of solutions produced with each MOEA.
- Comparison of the results with other multi-objective and traditional clustering algorithms.
- Evaluation of the impact of initial partitions in the multi-objective clustering.
- Qualitative analysis of the behavior of MOCLE, MOCK and Δ -MOCK according to dataset characteristics.

To achieve these objectives this study implemented the jMetal-based Multiobjective Cluster Ensemble (jMocle) which uses as core MOCLE integrated with other MOEAs available in the jMetal framework.

While jMocle enables the analysis related to the different MOEAs, to verify the impact of the initial population, we used the traditional clustering algorithms that are already part of the MOCLE technique and a modified version of MOCK adapted to receive the same sets of initial partition used in jMocle.

1.3 WORK ORGANIZATION

This work is organized as follows: Chapter 2 presents the main concepts and theoretical background related to clustering. Chapter 3 describes the multi-objective optimization problem and the multi-objective algorithms of interest. Chapter 4 provides an overview of works with multi-objective clustering, describing in more detail the MOCK and MOCLE techniques. Chapter 5 describes the proposed approach. Chapter 6 evaluates the proposed approach. Chapter 7 presents a data-driven analysis of the behavior of MOCLE, MOCK and Δ -MOCK. The conclusions are presented in Chapter 8.

2 CLUSTERING

This chapter presents the main contexts, formal definitions, and algorithms related to clustering that are relevant to this work. Section 2.1 presents the definition and taxonomy of clustering and some clustering algorithms. Section 2.2 provides the cluster ensemble definition and some of its techniques. Section 2.3 describes cluster validation and presents the Adjusted Rand index.

2.1 CLUSTERING DEFINITION

Clustering is a technique to group objects in such a way that objects inside the cluster are more similar to each other and more distinct in relation to objects outside [35]. They can be broadly divided into two main types [49, 47]: partitional and hierarchical methods. Besides these two methods density-based clustering [51] will also be explained further ahead.

Clustering algorithms rely on objective functions to optimize a clustering criterion. Three of these objective functions that will appear in this work are the overall deviation (*odv*) and intra-cluster variance (*var*) which expresses the compactness criterion and connectivity (*con*) which expresses the connectedness criterion [40].

The overall deviation is computed according to Equation 2.1, where π is a partition, $\mathbf{x}_i$ is an object belonging to cluster c_k , $\boldsymbol{\mu}_k$ is the centroid of cluster c_k , and d is the selected distance function.

$$odv(\pi) = \sum_{c_k \in \pi} \sum_{\mathbf{x}_i \in c_k} d(\mathbf{x}_i, \boldsymbol{\mu}_k), \quad (2.1)$$

The intra-cluster variance is conceptually equal to the overall deviation, yet it is used specifically, for example, in [36]. It is computed according to Equation 2.2, where π denotes a partition, n is the number of objects in the dataset, $\mathbf{x}_i$ is an object belonging to cluster c_k , $\boldsymbol{\mu}_k$ is the centroid of the cluster c_k , and $d(.,.)$ is the selected distance function.

$$var(\pi) = \frac{1}{n} \sum_{c_k \in \pi} \sum_{\mathbf{x}_i \in c_k} d(\mathbf{x}_i, \boldsymbol{\mu}_k)^2, \quad (2.2)$$

The connectivity is computed according to Equation 2.3, where π is a partition, N is the number of objects in the dataset, L is the parameter that determines the number of nearest neighbors that contributes to the connectivity, nn_{ij} is the j -th nearest neighbor of object $\mathbf{x}_i$, and c_k is a cluster that belongs to π .

$$con(\pi) = \sum_{i=1}^N \sum_{j=1}^L f(\mathbf{x}_i, nn_{ij}) \quad (2.3)$$

$$f(\mathbf{x}_i, nn_{ij}) = \begin{cases} \frac{1}{j}, & \text{if } \nexists c_k : \mathbf{x}_i, nn_{ij} \in c_k \\ 0, & \text{otherwise} \end{cases}$$

2.1.1 Partitional Clustering

Partitional clustering performs the identification of k - a number often provided by the user - desired clusters usually all at once. The result is a one-level partition of the dataset [6]. More

formally, given a set of objects $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, a (hard/crisp) partition of X in k clusters can be defined as $\pi = \{c_1, c_2, \dots, c_k\}$ with $k < n$, such that [76]:

1. $c_j \neq \emptyset, j = 1, \dots, k$
2. $\bigcup_{j=1}^k c_j = X$
3. $c_j \cap c_l = \emptyset, j, l = 1, \dots, k$ and $j \neq l$ (This requirement is relaxed, for example, in fuzzy partitioning techniques.)

Partitional clustering methods applying brute force are computationally impractical, therefore heuristics were created to search approximated solutions, relying in a criterion function as an important factor, and one of the most used criterion functions is the SSE (sum of squared errors) [76]. The SSE is presented in the Equation 2.4, Where k is the number of clusters, N is the number of instances in the dataset, $\gamma_{ij} = \begin{cases} 1 & \text{if } x_j \in \text{cluster } i \\ 0 & \text{otherwise} \end{cases}$, and μ_i is the centroid of cluster i . So SSE corresponds to the sum of all squared distances between cluster elements and the the centroid of its respective clusters.

$$SSE = \sum_{i=1}^k \sum_{j=1}^N \gamma_{ij} \|x_j - \mu_i\|^2 \quad (2.4)$$

The best-known algorithm that applies SSE is k -means [63]. A basic k -means for practical problems can be implemented following the steps ahead [76]:

1. Initialize the k clusters at random or utilizing previous knowledge of the problem. Calculate all the centroids.
2. Assign each object to the nearest centroid.
3. Recalculate the centroids based on the current partition.
4. Repeat 2-3 until there is no change in the clusters or until a stopping criterion.

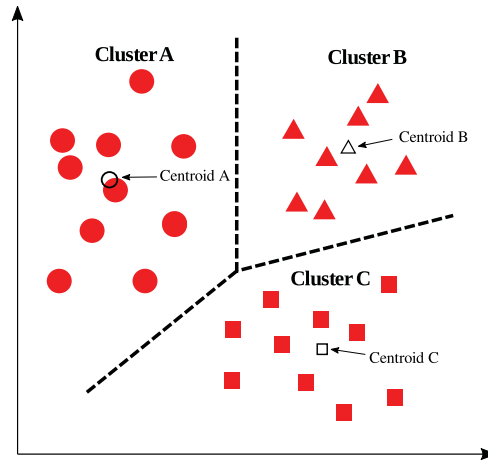


Figure 2.1: Illustration of k -means clustering.

The Figure 2.1 exemplifies a dataset partitioned into 3 clusters. The repetition of steps 2-3 aim to minimize the SSE until a stopping criterion is met, the stopping criterion can be the stabilization of the centroids (until the centroids stop moving). The SSE decreases as a function of k , so selecting a higher k tends to produce a lower SSE. Another known characteristic of k -means when using Euclidean distance is the identification of hyper-spherical clusters [49].

The necessity for a k selection can be considered a drawback as well as the sensitiveness to outliers and noise because these points can influence the calculation of mean value. This is mathematically proven in [13]. The complexity of k -means is $O(Nkd)$, where N is the number of instances, k the desired number of clusters and d the number of dimensions in the dataset [76].

2.1.2 Hierarchical Clustering

Hierarchical clustering algorithms are those who produce tree-like structures. The strategy used for hierarchical clustering can be agglomerative, where each object starts as one individual cluster and merge pairs of clusters through the hierarchy until reach the top with a single cluster containing all the objects. Otherwise a hierarchical clustering algorithm can be divisive, starting from the top and splitting pairs of clusters consecutively until reach the bottom with each object belonging to one cluster [76].

Hierarchical clustering algorithms data structures can be represented by a dendrogram [76].

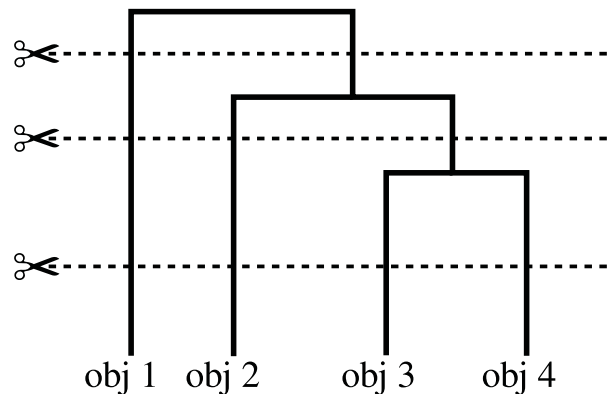


Figure 2.2: Example of dendrogram.

By cutting Figure 2.2 on delimited dotted lines it is possible to achieve numbers of cluster in a range from one to four clusters, with an intermediary interest zone of two or three clusters, given that is not necessary to have a heuristic to group data-points all in the same cluster or each one in individual clusters. An agglomerative algorithm would start from the bottom (4 clusters) while a divisive algorithm would start from the top (one cluster).

At every step of an agglomerative clustering algorithm is necessary to have some proximity measure between every pair of clusters to decide which pair to merge. That measure can be the proximity between the two closest objects of each cluster, that is the case of Single-link, or can be the distance between the centroids of each cluster in the case of Average-link [6].

A possible approach to agglomerative hierarchical clustering is Ward's method [74] which merges the pair of clusters that minimizes the increase in the total sum-of-squares, i.e., merge the pair of clusters that contribute with the smallest possible SSE, previously presented in the Equation 2.4.

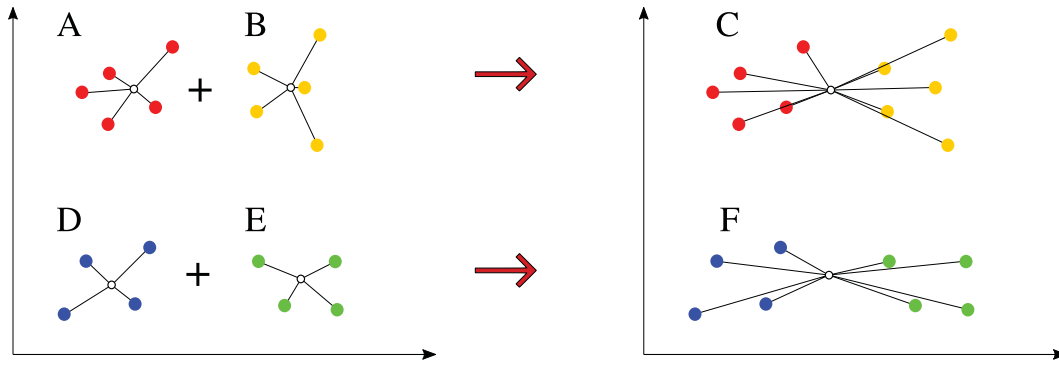


Figure 2.3: Illustration of Ward-linkage clustering

In Figure 2.3, the Ward-linkage verifies which is the smallest value, if it is the sum of the distances to the centroid in C compared to $A + B$ or in F compared to $D + E$.

Ward-linkage uses the same objective function as k -means, having the same bias towards the identification of hyper-spherical shapes. However, there are some differences between the two algorithms. Ward-linkage is deterministic because it doesn't have the random initialization of centroids, it starts from clusters of a single object and instead of reducing the SSE value, this value increases with each merge [74].

2.1.3 Density-based Clustering

Density-based clustering algorithms aim to identify clusters by taking into account a parameter of density threshold, considering clusters as the areas where calculated density among objects exceeds that parameter [51].

The SNN (Shared Nearest Neighbor) algorithm and a version of the DBSCAN (density-based spatial clustering of applications with noise) called HDBSCAN which arranges the density-based clustering into a hierarchy will be examples of two density-based algorithms used in this work.

The steps for SNN are [24]:

1. Compute the similarity matrix.
2. Compute the k -nearest neighbor list of each object.
3. Build a shared nearest neighbor graph linking two points if they have each other in their respective k nearest neighbor list.
4. Calculate the total strength (Equation available in [24]) of links coming out of each point in the graph. This step uses an algorithm input threshold that removes links with weight lower than that parameter.
5. Select the representative points, those with total link strength higher than a specific user input parameter.
6. Remove points having total link strength lower than a user input noise parameter.
7. Remove all links smaller than a merge threshold.
8. Form the clusters taking all connected components, being them representative points or non-representative points that still connected.

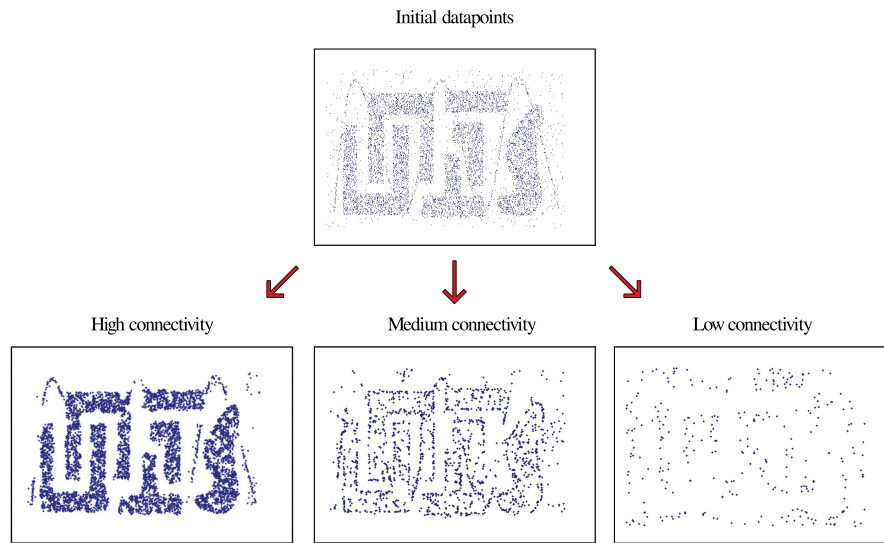


Figure 2.4: Example of SNN density classification. Adapted from [24]

Figure 2.4 illustrates a set of initial data-points and indicates high connectivity points, i.e., points with the highest number of strong connections in the neighborhood graph. The point with medium and low connectivity are also represented in the figure. The points with high connectivity are selected to be representative objects and the points of low connectivity are considered noise and are removed.

The SNN is capable to identify clusters with different shapes and sizes even in the presence of outliers and noise and works well with high dimensional data. The complexity of SNN is $O(N^2)$ [24].

A density-based approach clustering algorithm proposed by [12] called HDBSCAN is formulated around the traditional DBSCAN [25] and capable of representing DBSCAN-like structures into a hierarchy and extract a simplified tree with significant structures from it. This approach can overcome limitations present in traditional density-based algorithms as the identification of varying density structures or nested clusters.

The HDBSCAN steps are [64]:

1. Transform the space according to the density/sparsity.
2. Build the minimum spanning tree of the distance weighted graph.
3. Construct a cluster hierarchy of connected components.
4. Condense the cluster hierarchy based on minimum cluster size.
5. Extract the stable clusters from the condensed tree.

During the first step in order to find clusters areas of high density among noise, the HDBSCAN algorithm establishes a metric of distance between a point and its k -th nearest neighbor, called core distance. Then the algorithm pushes away points of low density (high core distance) using the mutual reachability distance metric (Equation available in [12]).

By using a threshold to remove edges with higher weights and varying the threshold levels it is possible to build a hierarchy from completely connected graph to a completely disconnected. So the second step builds a minimum spanning tree graph with data points as

vertices and edges weights corresponding to the values of mutual reachability distances, in that way any removal of edges will cause a disconnection of components and formation of separate clusters.

The third step sorts the edges of the tree by increasing order of distance and iterates merging each edge to create a new cluster. The result for step 3 is a dendrogram similar to Figure 2.2 and at this point what a traditional DBSCAN algorithm would do is to cut the dendrogram at some user input number of clusters to produce the result partition.

In the fourth step to extract clusters with variable density, HDBSCAN condenses the cluster hierarchy into a smaller structure. To make this possible the algorithm uses a minimum cluster size input parameter to walk the dendrogram checking at each split if the new cluster has fewer points than the minimum cluster size. If the split cluster has fewer points than the input parameter it will be considered as "points falling out of parent cluster" and the information on which point and at which distance it felt will be stored in the parent node. If the split cluster has at least the minimum cluster size value than it will be considered a genuine cluster and the split validated. An example of tree dendrogram condensation is presented in Figure 2.5:

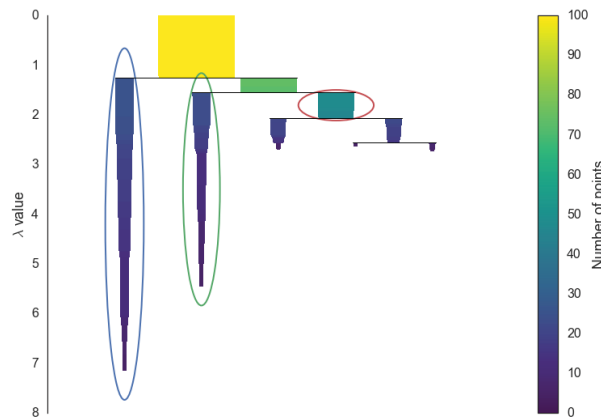


Figure 2.5: Example of condensed dendrogram. [64]

The last step consists of selecting the clusters with the largest area in the condensed dendrogram with the restriction that after a selection of a cluster its descendants cannot be further selected.

2.2 CLUSTER ENSEMBLE

Traditional clustering techniques such as those presented in Section 2.1 face some difficulties when confronting datasets with more than one level of structures or heterogeneous cluster shapes as the ones presented in the Figure 2.6.

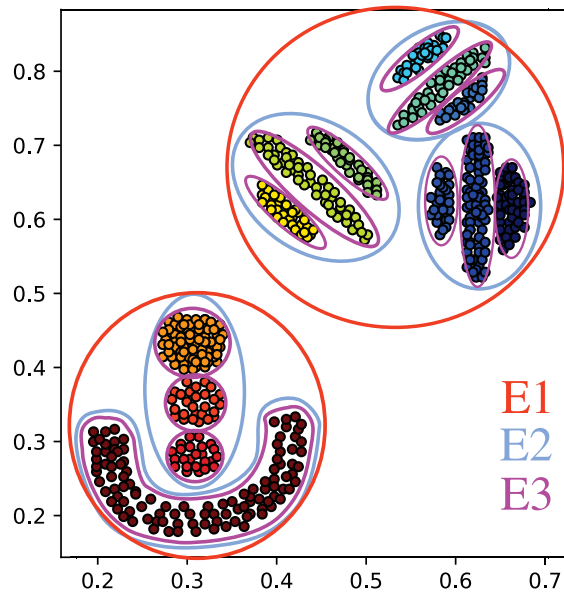


Figure 2.6: Dataset with multiple structures. Adapted from [26].

Algorithms as k -means and Ward-linkage, with a bias to identify hyper-spherical shapes, would easily find the E1 structure but are not able to identify structure E2 which has a smile shaped cluster.

Partitions obtained by different clustering algorithms can capture different aspects of the data. The cluster ensemble technique was developed to address the problem of traditional clustering algorithms having a bias for determined shapes. The problem of the cluster ensemble is to find a new partition that summarizes the information obtained by other partitions in a dataset [70].

The method of cluster ensemble consists of two steps, the first step is to generate a diverse set of base partitions and during the second step, those partitions are combined using a consensus function defined as a function that maps a set of partitions to an integrated one called consensus partition [68].

During the first step, for the base partitions contribute with useful information for the consensus partition is necessary an adequate selection of clustering algorithms that provide diversity. Some strategies can be applied such as:

- To generate the base partitions using different traditional clustering algorithms [68].
- To execute an algorithm many times choosing different parameters [70].
- To evaluate different datasets employing re-sampling or by projection into a smaller feature space [32].

Since consensus functions are based in heuristics the same way as clustering criteria, there are many approaches in which they can be based:

Co-association-based functions: Estimates the similarity between a pair of objects by counting the number of clusters they share considering the base partitions of the ensemble and storing that number into a matrix.

Hypergraph/Graph-based functions: Represents the base partitions as hyper-graphs or graphs and uses some efficient graph or hyper-graph partitioning heuristic to partition the graph or hyper-graph.

Relabeling-based functions: Uses a voting procedure to assign objects from the base partition to the consensus partition according to the object labels. Since in the unsupervised classification the labels of each object are not known usually a reference partition is chosen to guide another partitions re-labeling.

The graph-based techniques MCLA (Meta-Clustering algorithm) [68], and HBGF (Hybrid Bipartite Graph Formulation) [32] and the co-association-based CSPA (Cluster-based Similarity Partitioning Algorithm) [68] are cluster ensemble methods used in this work and explained below:

2.2.1 MCLA

The MCLA (Meta-Clustering algorithm) [68] uses the shared mutual information between individual solutions to formally define the ensemble. The main steps of MCLA are:

- Build a meta-graph: each hyper-edge belonging to the initial hype-graph constitutes a vertex in the meta-graph. Since each vertex corresponds to a cluster, then the edge weight is the ratio between the intersection by the union of both vertices.
- Cluster hyper-edges: the clustering of hyper-edges, which correspond to clusters, is done by partitioning the meta-graph in k balanced meta-clusters. Since each vertex is weighted proportionally to the size of the cluster the balancing of weights tries to ensure that the sum of weights is the same in each meta cluster.
- Collapse meta-clusters: To transform the hyper-edges of each meta cluster into a single meta-hyper-edge. Since each object can belong to more than one meta-cluster, the level of association is calculated by averaging all indicator vector of a particular meta-cluster and storing that value in an association vector of the meta-hyper-edge.
- Compete for objects: Each object is signed to the meta-cluster which its most associated. Ties are randomly resolved. Since is not guaranteed that each meta-cluster will have at least one object, the consensus partition will have a maximum of k cluster.

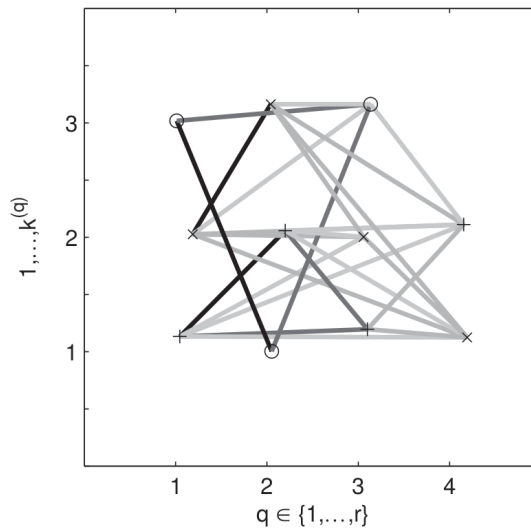


Figure 2.7: Illustration of MCLA. [68]

In the Figure 2.7 the symbols $\circ$, $\times$ and $+$ indicate the meta-clusters. Darker edges have a higher weight. Assuming that after collapsing $\circ$ hyper-edges the new meta-hyper-edge contains the vertices/objects $\{v5, v6, v7\}$ with an association vector $\{0, 0, 0, 1/3, 1, 1\}$ then the meta cluster $\circ$ wins the competition for objects $v6$ and $v7$, a cluster is generated and the same process happens with the other meta-clusters.

2.2.2 HBGF

The HBGF (Hybrid Bipartite Graph Formulation) [32], builds a bipartite graph from an initial partition set modeling simultaneously objects and clusters as vertices of the graph, then any graph partition technique can be applied to partition the graph into a desired number of clusters. The algorithm steps are the following:

- Build a graph $G = (V, W)$ with V representing a set of vertex and W is a non-negative and symmetric $|V| \times |V|$ similarity matrix characterizing the similarity between each pair of vertices. G is built from the base partition according to:
 - $V = V^C \cup V^O$, with the vertex in V^C representing the clusters in the initial partition set, and the vertex in V^O representing the objects in the dataset.
 - $W(i, j) = 0$ if $i, j \in V^C$ or $i, j \in V^O$, otherwise if the object i belongs to cluster j , then $W(i, j) = 1$, else $w(i, j) = 0$.
- G is partitioned into k groups defined by the user. The software METIS is used by [32]. As a result, we have π^F , the consensus partition.

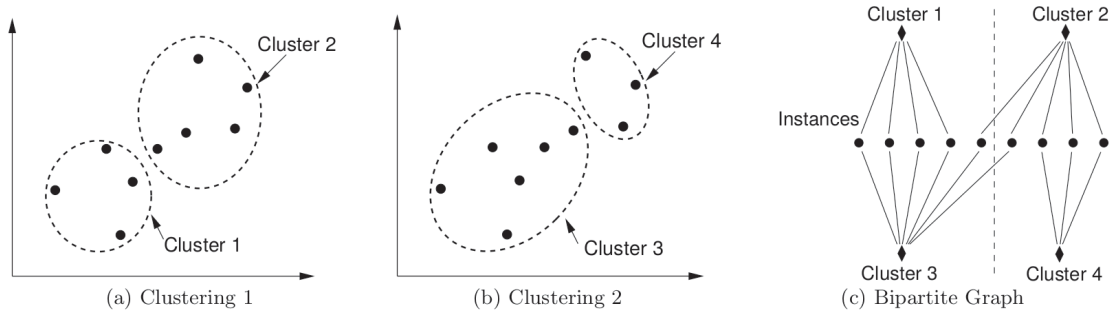


Figure 2.8: Illustration of HBGF. [32]

In the Figure 2.8, (c) have both the clusters from (a) and (b) represented as vertices as well as the instances, the dotted line in (c) indicates a partition in the graph originating two clusters.

2.2.3 CSPA

The CSPA (Cluster-based Similarity Partitioning Algorithm) [68] has a simple heuristic. It creates a $N \times N$ binary similarity matrix for each partition and averages them into an overall similarity matrix. Figure 2.9 illustrates the construction of the similarity matrix.

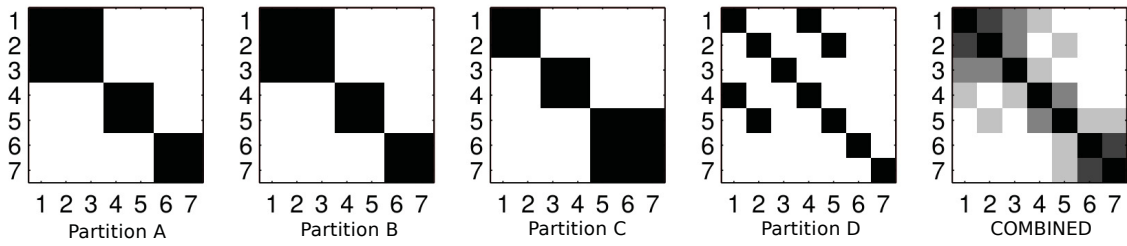


Figure 2.9: CSPA similarity matrix construction. Adapted from [68]

In the next step a similarity graph is constructed with vertex being objects and edges weights being similarity values. Finally applies the graph partitioning tool METIS [50] in the graph to obtain the consensus partition.

Traditional clustering algorithms or cluster ensemble techniques will produce result partitions that have to be evaluated to measure the quality of clusters, as well as the efficiency of the technique.

2.3 CLUSTER VALIDATION

The validation of clusters is made by indexes that quantify its quality indicating that the identified structures do not occur by randomness. Three types of criteria can be used to validate a cluster [48]:

- External criteria lie in prior knowledge of structures in the dataset to evaluate the given partitions generated by an algorithm in contrast with labeled data or partitions built by specialists.
- Internal criteria establish the validity of a cluster-based exclusively in the dataset itself, for example, how much a cluster is justified by the means of the proximity matrix.
- Relative criteria measure qualitatively which algorithm fits the data better for a case, or which algorithm is more stable for another case.

Internal and external criteria aim to specify how much the results obtained confirm a pre-specified hypothesis. This is done through statistical tests that indicate whether the value of the index is significantly large or small and this requires the establishment of a base population. The same indices can be used in internal or external criteria, the difference between the two is that in the external criteria considers already know structures, whereas in internal criteria only current data is taken into account [48].

2.3.1 Adjusted Rand Index

The Adjusted Rand index (ARI), is an external index developed by 46, is one of the most widely used external validation indices for comparing clustering algorithms. It does not suffer interference from the number of groups as in other external validation indexes [48].

The Rand Index is defined as $R = \frac{a + b}{\binom{n}{2}}$. With a representing the number of times a pair of object belongs to the same cluster in two different partitions, b representing the number of times a pair of objects belong to different clusters in two different partitions and $\binom{n}{2}$ is the number of unordered pairs in a set of n elements, with n being the size of the dataset [48].

The Adjusted Rand is a corrected-for-chance version of the Rand index. To correct-for-chance means that the statistical index is normalized to return 0 if its value is selected by chance (randomly) and 1 if it is a perfect match. [48].

$$\frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex} \quad (2.5)$$

Based on Equation 2.5 the *Max Index* is replaced by 1 and the *Expected Index* is resolved by hypergeometric distribution, i.e., partitions are chosen randomly such that the number of objects in the clusters is fixed, resulting in the ARI.

$$ARI = \frac{\sum_{i=1}^{k^a} \sum_{j=1}^{k^b} \binom{|c_i^a \cap c_j^b|}{2} - \left[\sum_{i=1}^{k^a} \binom{|c_i^a|}{2} \sum_{j=1}^{k^b} \binom{|c_j^b|}{2} \right] / \binom{n}{2}}{\left[\sum_{i=1}^{k^a} \binom{|c_i^a|}{2} \sum_{j=1}^{k^b} \binom{|c_j^b|}{2} \right] / 2 - \left[\sum_{i=1}^{k^a} \binom{|c_i^a|}{2} \sum_{j=1}^{k^b} \binom{|c_j^b|}{2} \right] / \binom{n}{2}} \quad (2.6)$$

In the Equation 2.6 k^a and k^b are the number of clusters of partitions π^a and π^b , c_i^a corresponds to the i -th cluster from partition π^a , c_j^b is the j -th cluster from π^b and n is the number of objects of the data-set. For example, given a partition π^a generated with a clustering algorithm for a dataset and a partition π^b representing the underlying structure of the data (true partition/ground truth), this measure determines the similarity between π^a and π^b by analyzing the pairwise co-assignment of points between the two partitions. For partitions $\pi^a = [0,0,1,1]$ and $\pi^b = [1,1,0,0]$ we have an ARI = 1, for partitions $\pi^a = [0,0,1,1]$ and $\pi^b = [0,1,2,3]$ we have an ARI = 0. It is possible to find negative values as in the case of $\pi^a = [0,0,1,1]$ and $\pi^b = [0,1,0,1]$ we have the ARI value = -0.5. The upper bound of ARI is 1, indicating a perfect agreement between the partitions, and values near 0 or negatives corresponding to cluster agreement found by chance [46].

2.4 DISCUSSION

This chapter focused on important concepts for the course of this work and showed some of the various approaches and taxonomies related to clustering. Simple algorithms such as k -means prove to be efficient for a variety of problems but are not able to identify every type of structure present in the data. Cluster ensembles try to overcome this type of problem by combining different partitions.

k -means proves to be a robust algorithm for the formation of initial partitions, despite having biases for clusters with a hyper-spherical format. The Ward-linkage hierarchical algorithm also has a bias to identify clusters with hyper-spherical formats, so to balance the composition of the initial partitions were included the SNN and HDBSCAN, two density-based algorithms capable of identifying arbitrary formats. Table 2.1 summarizes the characteristics of the selected algorithms.

Table 2.1: Summary comparison of clustering algorithms

Algorithm	Clustering Criteria	Cluster Shapes	Noise Resistant
<i>k</i> -means	compactness	hyper-spherical and compacts	no
Ward-linkage	compactness	hyper-spherical and compacts	no
SNN	connectedness	arbitrary shapes, sizes and density	yes
HDBSCAN	connectedness	arbitrary shapes and sizes	yes

The next chapter talks about the concept of multi-objective optimization and presents the algorithms for this purpose used in this work.

3 MULTI-OBJECTIVE OPTIMIZATION

This chapter describes the problem of multi-objective optimization, a necessary concept for the understanding of the approach proposed in this study. Further, it will present the multi-objective algorithms used in this work. Section 3.1 presents the multi-objective problem and the Pareto front definitions. Section 3.2 describes the taxonomy and some multi-objective algorithms of interest.

3.1 MULTI-OBJECTIVE OPTIMIZATION

A multi-objective optimization problem can be stated by the following Equation 3.1 [14, 80].

$$\text{minimize } F(x) = (f_1(x), \dots, f_m(x)), \quad x \in \Omega \quad (3.1)$$

Subject to:

$$g_i(x) \geq 0 \quad i = 1, 2, \dots, p \quad (3.2)$$

$$h_i(x) = 0 \quad i = 1, 2, \dots, q \quad (3.3)$$

Where m represents the total number of objective functions, p are inequality constraints, q are equality constraints, Ω is the *decision space*, $F : \Omega \rightarrow R^m$ consists of m real valued objective functions and R^m is called *objective space*. The *attainable objective set* is defined as the set $\{F(x) | x \in \Omega\}$.

Generally, the objectives of multi-objective optimization are contradictory and the improvement of one leads to the worsening of the other, so there should be a balance between them. The best trade-offs between these objectives can be defined through Pareto optimization (80).

3.1.1 Pareto Dominance

Being $u, v \in R^m$, it is said that u dominates v if and only if $u_i \geq v_i$ for all $i \in \{1, \dots, m\}$ e $u_j > v_j$ for at least one index $j \in \{1, \dots, m\}$, so u dominates v if u is as good as v for all objectives, and is better than v for at least one objective [80, 61]. The Figure 3.1 illustrates a minimization problem with two objectives.

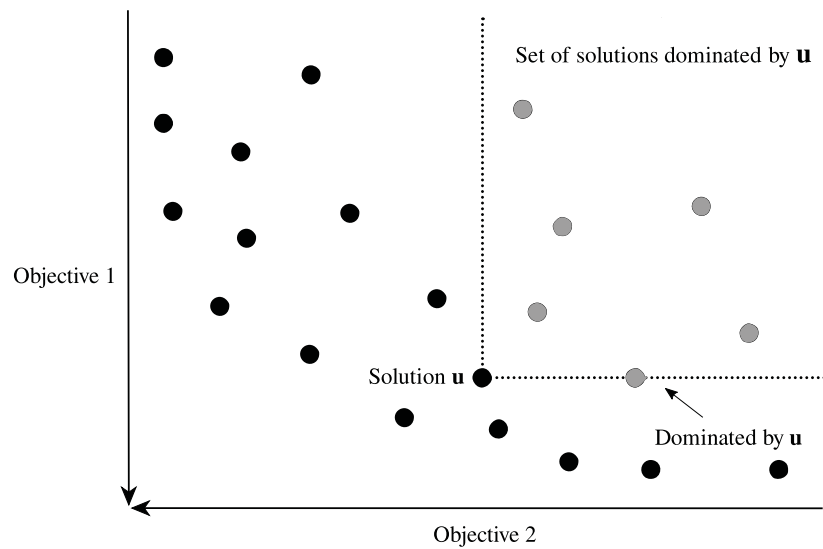


Figure 3.1: Illustration of Pareto dominance. Adapted from [61]

A point x^* is Pareto optimum if there is no other point in the solution space that dominates it. The set containing all the Pareto optimal is called the Pareto set, and the target function values for the Pareto optimal set solutions form the Pareto front, represented in Figure 3.2.

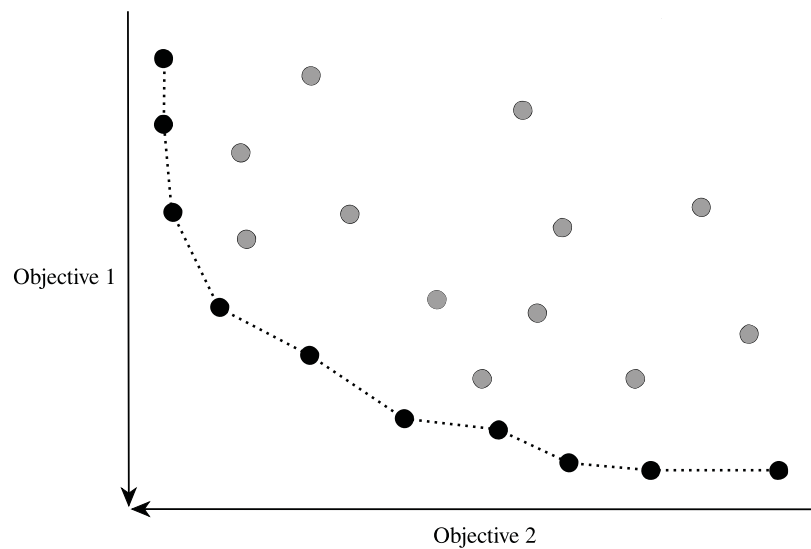


Figure 3.2: Illustration of a Pareto non-dominated front. Adapted from [61]

Connected by the dotted line are the points that integrate the Pareto front. These points dominate all other solutions of the population and represent different trade-offs between the two objectives.

3.2 MULTI-OBJECTIVE EVOLUTIONARY ALGORITHMS

The literature presents a great diversity of MOEAs (Multi-Objective Evolutionary Algorithms) [84], algorithms for solving multi-objective problems using a population and natural evolutionary mechanisms.

MOEAs are based on the Genetic Algorithms [45] which simulate the natural evolution process. To evolve solutions they use the following basic steps [11]:

1. **Initialization:** An initial population of solutions is created randomly or with the contribution of problem-specific information.
2. **Evaluation:** The solutions of the initial population are evaluated as well as the solutions created in the offspring populations. The evaluation measures the quality of the solution considering the objective values defined by objective functions, the quality of solutions is known as *fitness*.
3. **Selection:** This mechanism imposes the survival-of-the-fittest by allocating more copies for solutions with better fitness values. Many selection operators are proposed in the literature [11] such as roulette-wheel, ranking selection, and tournament selection.
4. **Recombination:** The recombination or *crossover* operator combines two or more solutions previously selected to generate a possibly better new solution. There are several options for crossover operators, many of them are created specifically for some problem.
5. **Mutation:** It is a procedure that operates individually in solutions performing a random-walk and making modifications. The literature presents several mutation operators.
6. **Replacement:** The initial or previous population is replaced by an offspring population created using selection, recombination and mutation. MOEAs can use different replacement methods such as elitist, generation-wise, and steady-state.
7. Repeat steps 2-6 until a stopping criterion is met.

A recurrent necessity in multiobjective optimization is the estimation of an objective vector with the worst value for each objective function, known as the nadir vector. This is a step that appears in multiple MOEAs and among the reasons for its application are [19]:

- It is used with the objective vector, which is a point representing the best possible values for each objective function. The nadir and ideal vector can be used as bounds to normalize objective functions, usually when these functions present different magnitudes.
- Also together with the ideal vector it can provide the range of each objective and help the decision-maker to adjust expectation, use specific information or techniques according to the region of the front in which the solutions are and visually compare different solutions.

Figure 3.3 illustrates in a minimization problem the evolution of solutions starting from the initial population.

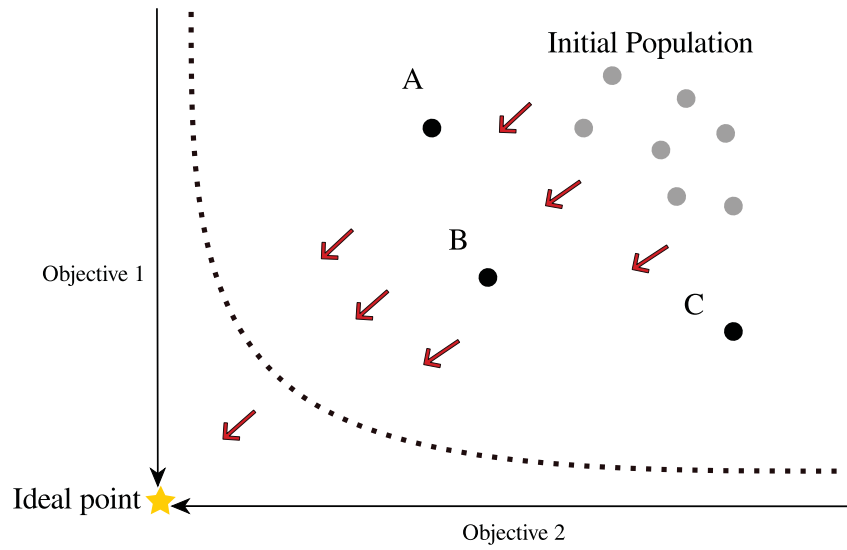


Figure 3.3: Illustration of the MOEA evolution of solutions in a minimization problem.

In Figure 3.3, solution A was more successful in minimizing objective 2, solution C minimized further objective 1, and solution B had a good balance minimizing both. The ideal point indicates the perfect minimization of all objectives.

MOEAs can be divided into several classes according to the type of strategy used to solve the optimization problem [56, 69].

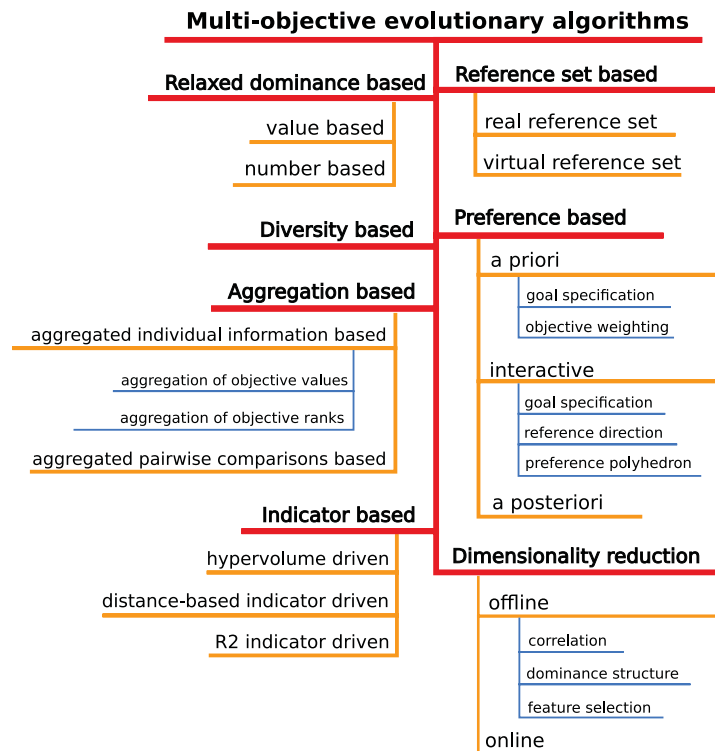


Figure 3.4: Classification of multi-objective evolutionary algorithms. Adapted from [55]

The categories presented in Figure 3.4 are described next:

Pareto-based: Are algorithms that perform evolution based on Pareto dominance relation using as a criterion to distinguish solutions during the evolutionary process. The most common algorithm in this category is the NSGA-II [20].

Relaxed dominance-based: Modify the Pareto dominance ratio to increase the selection pressure towards the Pareto front. Algorithms such as dominance- ϵ and diffuse dominance use this approach [53, 43]. Also, the algorithm θ -DEA (θ dominance-based evolutionary algorithm) [79] develops and uses the relationship of dominance θ .

Reference set-based: A set of solutions is used to guide the search process and to measure the quality of the populations [55].

Preference-based: When the multi-objective problem has many dimensions the number of solutions tends to increase, then it is rational that a subset of the Pareto front is defined by the user according to his preference. The algorithms of this class can be identified as a priori, interactive or a posteriori. In a priori algorithms, the preference information is stipulated before the search. In a posteriori algorithms, the preference information is defined after the search. In interactive algorithms, the user informs the preference interactively. The PICEA-g posterior algorithm uses user-defined preferences, achieving competitive performance as the number of objectives increases [73].

Diversity-based: Focus on the criterion related to maintaining the diversity of solutions. The GrEA (grid-based evolutionary algorithm) [77] introduces three criteria aimed at solution diversity. The KnEA (knee point driven evolutionary algorithm) uses a secondary selection scheme to accelerate convergence while maintaining diversity [82]. The reference point strategy is used by algorithms such as NSGA-III to attract solutions to these diverse points [18].

Decomposition-based: Decompose the multi-objective optimization problem into several mono-objective problems. The MOEA/D was the first algorithm to implement this optimization strategy [80]. Three functions known as scalarization or aggregation functions are the most common to be used as a decomposition method. They are PBI (penalty-based boundary intersection), Tchebycheff and Weight Sum.

PBI can be defined as [57]:

$$\text{minimize } g^{pbi}(x|\lambda, z^*) = d_1 + \theta d_2 \quad \text{subject to } x \in \Omega \quad (3.4)$$

where:

$$d_1 = \frac{\|(F(x) - z^*)^T \lambda\|}{\|\lambda\|} \quad (3.5)$$

$$d_2 = \|F(x) - (z^* + d_1 \frac{\lambda}{\|\lambda\|})\| \quad (3.6)$$

Where λ is a weight vector, $z^* = (z_1^*, \dots, z_m^*)^T$ is the ideal objective vector with $z_i^* < \min_{x \in \Omega} f_i(x), i \in \{1, \dots, m\}, \theta \geq 0$ is a user defined penalty parameter. Figure 3.5 illustrates the PBI approach.

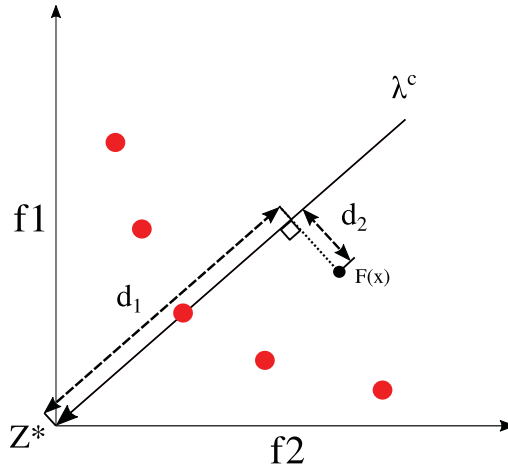


Figure 3.5: Illustration of PBI decomposition

The objective of PBI is to push $F(x)$ towards the Pareto front. For this, parameter θ in the Equation 3.4 controls the balance between the convergence represented by d_1 and the diversity represented by d_2 .

Equation 3.7 presents the Tchebycheff function [80]:

$$\text{minimize } g^{Te}(x|\lambda, z^*) = \min_{1 \leq i \leq m} \{\lambda_i |f_i(x) - z_i^*|\} \quad \text{subject to } x \in \Omega \quad (3.7)$$

Where $z^* = (z_1^*, \dots, z_m^*)^T$ is the reference point, i.e., $z_i^* = \min\{f_i(x) | x \in \Omega\}$ for each $i = 1, \dots, m$, with Ω being the decision space. For each Pareto-optimal point x^* there is a weight vector λ , such that x^* is the optimal solution for 3.7. Each optimal solution for Equation 3.7 is an optimal solution for the multi-objective optimization problem presented in 3.1.

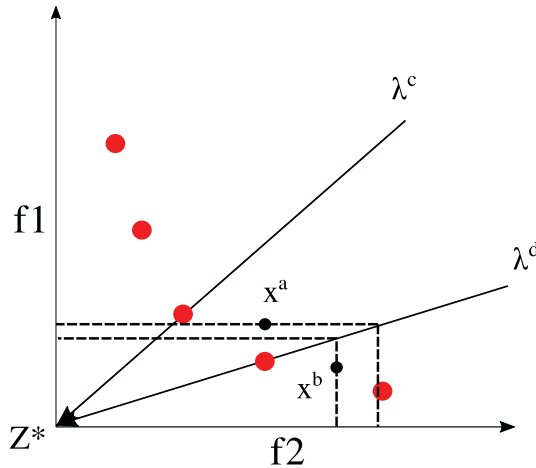


Figure 3.6: Illustration of Tchebycheff decomposition

In the Figure 3.6. For the weight vector λ^d , solution x^b has a fitness value greater than x^a .

Indicator-based: Use the value of a performance indicator to guide the search process. The IBEA (indicator-based evolutionary algorithm) [86] is a pioneer in this strategy and works with the binary quality indicator. In view of the problem of maintaining the

diversity that affects the IBEA, other algorithms such as SMS-EMOA and HypE use the Hypervolume indicator, which balances convergence and diversity appropriately [7, 5].

Dimensionality reduction: Problems with a higher number of objectives have more expensive running times and face difficulties in obtaining high-quality sets of solutions. These problems have been addressed by strategies like PCA (Principal Component Analysis) [21], or an exact and an approximate way to reduce the number of objectives working with conflicting objective sets [10]

Hybrids: They use more than one strategy such as the MOEA/DD algorithm, which combines dominance and decomposition [57]. The algorithms Two_Arch2, HEA-DP (hybrid evolutionary algorithm with dual populations) and BCE (bi-criterion evolution) maintain two populations that are updated separately in order to evolve two criteria separately [72, 83, 60].

Next, we describe the multi-objective algorithms used in the proposed system.

3.2.1 NSGA-II

According to [20], The NSGA-II (nondominated sorting genetic algorithm II) addresses multi-objective optimization through an evolutionary algorithm based on nondominated sorting. The selection of solutions prioritizes lower Pareto rankings. The Algorithm 1 represents NSGA-II which is detailed in the numbered steps in the sequence:

Algorithm 1 NSGA-II. Adapted from [15]:

Input: $N, g, f_k(x)$

- 1: Initialize Population P_0
- 2: Evaluate Objective Values of P_0
- 3: Assign Rank Based on Pareto dominance - sort to P_0
- 4: Generate Child Population Q_0
- 5: **begin**
- 6: Binary Tournament Selection
- 7: Recombination and Mutation
- 8: Evaluate Objective Values of Q_0
- 9: **end**
- 10: **for** $i=1$ to g **do**
- 11: $R_t = P_t \cup Q_t$
- 12: **for** each $x \in R_t$ **do**
- 13: Assign Rank Based on Pareto dominance-sort
- 14: **end for**
- 15: Generate sets of nondominated (F_t) vectors along PF known
- 16: Determine crowding distance between points on each front (F_t)
- 17: Select points on the lower front and are outside a crowding distance to generate P_{t+1}
- 18: Generate Offspring Population Q_{t+1}
- 19: **begin**
- 20: Binary Tournament Selection
- 21: Recombination and Mutation
- 22: Evaluate Objective Values of Q_{t+1}
- 23: **end**
- 24: **end for**

1. The algorithm input N is the population size, g is the number of generations and $f_k(x)$ the objective functions.
2. The lines 1-3 generate a random initial population P_0 , its objective values are evaluated and the population is ordered according to the dominance rank.
3. In lines 4-9 the selection, recombination, and mutation operators are applied in P_0 to generate the offspring population Q_0 .
4. The algorithm starts a loop for the number of generations g or until another stopping criterion is met in line 10.
5. During the main loop the Parent and Offspring populations are combined into R_t (line 11). The combined pool is split into non-dominated levels according to individual non-dominated ranks (lines 12-15).
6. Create a new population P_{t+1} by selecting members of the best fronts and crowding distances (line 17) and apply the genetic operators in P_{t+1} to create a new offspring population. If the stopping criterion is not met then the loop continues from these new populations.

To determine the solution after ranking according to the Pareto non-dominance rankings, if the population in the best F_1 ranking is less than N , the other members of the P_{t+1} population will be selected from the subsequent rankings, otherwise, a crowding distance comparison is used to select the best individuals and fill the P_{t+1} population. The selection, crossover and mutation operators are applied to the population P_{t+1} thus creating the new population Q_{t+1} of size N [20].

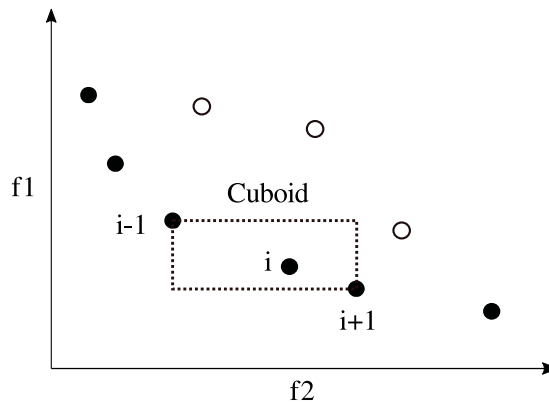


Figure 3.7: Crowding comparison. Adapted from [20]

Crowding distance is calculated as the area of the cube formed by the two nearest neighbors, illustrated in Figure 3.7.

3.2.2 NSGA-III

The NSGA-III, according to [18], renews the previous version of the algorithm by focusing on members of the population closest to predefined reference points that serve to ensure diversity

in solutions and by replacing the crowding comparison operator for a strategy based on such reference points.

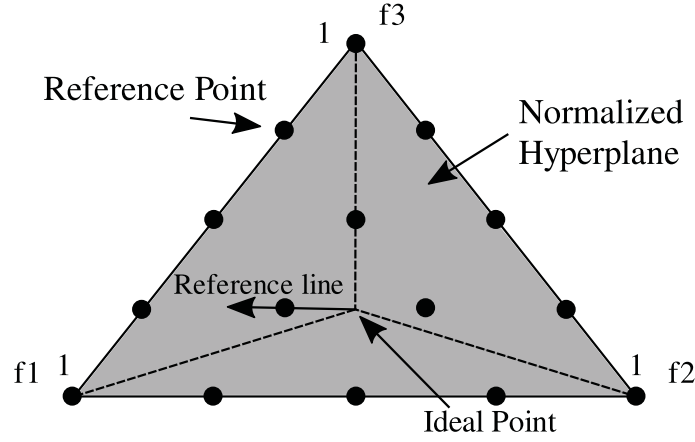


Figure 3.8: Reference Point Structure. Adapted from [18]

As shown in Figure 3.8, a number of reference points is predefined through Das and Dennis's method [17], or can be supplied by the user. These points are used to obtain a good distribution over or near the Pareto border.

The pseudo-code of NSGA-III is presented in the Algorithm 2:

Algorithm 2 NSGA-III. Adapted from [18]

Input: $N, g, f_k(x)$

```

1: ... similar to NSGA-II initialization
2: for  $i=1$  to  $g$  do
3:    $R_t = P_t \cup Q_t$ 
4:   for each  $x \in R_t$  do
5:     Assign Rank Based on Pareto dominance - sort
6:   end for
7:   Generate sets of nondominated ( $F_t$ ) vectors along PF known
8:   repeat
9:      $S_t = S_t \cup F_t$     $i = i + 1$ 
10:  until  $|S_t| \geq N$ 
11:  if  $|S_t| = N$  then
12:     $P_{t+1}$ , break
13:  else
14:     $P_{t+1} = \cup_{j=1}^{l-1} F_j$ 
15:    Points to be chosen from  $F_l : K = N - |P_{t+1}|$ 
16:    Normalize objectives and create reference set  $Z^r$  :  $Normalize(f^n, S_t, Z^r, Z^s, Z^a)$ 
17:    Associate each member  $s$  of  $S_t$  with a reference point:  $[\pi(s), d(s)] = Associate(S_t, Z^r)$  //
     $\pi(s)$  : closest reference point,  $d$  : distance between  $s$  and  $\pi(s)$ 
18:    Compute niche count of reference point  $j \in Z^r$  :  $\rho_j = \sum_{s \in S_t/F_l} ((\pi(s) = j) ? 1 : 0)$ 
19:    Chose  $K$  members one at a time from  $F_l$  to construct  $P_{t+1}$ :  $Niching(K, \rho_j, \pi, d, Z^r, F_l, P_{t+1})$ 
20:  end if
21:  ...similar to NSGA-II generation of  $Q_t + 1$  (Offspring population)
22: end for

```

The NSGA-III has a basic framework similar to NSGA-II. Here is the explanation of the algorithm focusing on the main changes:

1. Line 1 initializes and computes the non-dominated fronts similarly as NSGA-II (see Algorithm 1).
2. Line 16 introduces the differences between NSGA-III and NSGA-II. Each non-dominance level is selected, one at a time, to make up a new S_l population, starting with F_1 until the size of S_l equals N . Since the last level included was l -th, all solutions from the $l + 1$ level onwards will be rejected. In most cases, the l -th level is partially accepted. In those cases, niche procedure will choose solutions that maximize the diversity of the l -th border.
3. From Line 29 the processes of generating a new offspring population are similar to NSGA-II.

3.2.3 SPEA2

According to [87], to calculate the individual's fitness the algorithm SPEA2 takes into account how many individuals it dominates and by how many it is dominated. To estimate density SPEA2 calculates the distance to all neighboring individuals whether they are in the ranking belonging to the individual or in the rest of the population. The algorithm keeps the best Pareto front members in a repository.

The Strength Pareto Evolutionary Algorithm (SPEA) serves as the basis for SPEA2, it uses a regular population and a repository. Initially, all nondominated members of the population are copied into the repository, if the size of the repository exceeds a limit the elimination of members in excess occurs through a clustering technique that preserves the characteristics of the nondominated front, then fitness values are calculated for the members of the archive and the population in two steps:

Step 1) Each individual in the archive receives a $S(i) \in [0.1]$ strength value which also represents their $F(i)$ fitness value. $S(i)$ is equal to the number of j members of the population that are dominated by or equal to i in the target values, divided by the size of the population plus one.

Step 2) The fitness $F(j)$ of an individual j belonging to the population is calculated by adding the $S(i)$ strength values of all the i archive members who dominate or equal j , adding one at the end.

The next stage is the crossover, where through a binary tournament the individuals in the archive and population are selected. Since fitness should be minimized, the members in the archive are more likely to be selected than the population members which receive a +1 during the fitness evaluation. Finally, recombination and mutation are performed and the old population is replaced by the new population.

The pseudo-code of the SPEA2 is presented in the Algorithm 3. SPEA2 receive as input the population size, N , archive size, $\bar{N}$, and number of generations, g .

Unlike the SPEA, fitness is determined by taking into account the strength of the dominant members in both the archive and the population, and not only that of the archive. In cases of a tie after the fitness calculation, adapted density information from the k -th nearest neighbor method is used, where the inverse of the distance to the k -th nearest neighbor is taken into account.

$$D(i) = \frac{1}{\sigma_i^k + 2} \quad (3.8)$$

Algorithm 3 SPEA2. Adapted from [87]:

Input: $N, \bar{N}, g$

- 1: Initialize Population P_0
- 2: Create a repository $\bar{P}$
- 3: **for** each individual in P_t and $\bar{P}_t$ **do**
- 4: Evaluate the fitness of the individual
- 5: **end for**
- 6: $\bar{P}_{t+1} \leftarrow$ nondominated individuals in P_t and $\bar{P}_t$
- 7: **if** $|\bar{P}_{t+1}| > \bar{N}$ **then**
- 8: reduce $\bar{P}_{t+1}$ by means of the truncation operator
- 9: **else if** $|\bar{P}_{t+1}| < \bar{N}$ **then**
- 10: fill $\bar{P}_{t+1}$ with nondominated solutions in P_t and $\bar{P}_t$
- 11: **end if**
- 12: **if** stopping criterion is satisfied **then**
- 13: $A \leftarrow$ nondominated individuals in $\bar{P}_{t+1}$
- 14: **return** A
- 15: **end if**
- 16: Perform binary tournament selection with replacement in $\bar{P}_{t+1}$
- 17: $P_{t+1} \leftarrow$ Apply crossover and mutation in P_{t+1} , go to line 3

σ_i^k indicates the distance between the k -th element and i . The value 2 is added to the denominator to ensure that the result is greater than zero and less than 1. Finally, the final fitness calculation can be established as Equation 3.9:

$$F(i) = R(i) + D(i) \quad (3.9)$$

During the selection phase, the first step is to copy all unmanaged individuals from the current generation to the new generation repository according to the Equation 3.10:

$$\bar{P}_{t+1} = \{i | i \in P_t + \bar{P}_t \wedge F(i) < 1\} \quad (3.10)$$

In case the Pareto front of the current population has the exact size of the repository then the selection process is completed, otherwise two actions may occur. In the first case, if the Pareto front is smaller than the repository, individuals will be copied from the previous repository and population sorted according to their fitness values, until the required number of individuals is obtained. In the second case, if the number of individuals on the Pareto front is excessive, the truncation procedure will be triggered.

During the truncation procedure, the individual who has the shortest distance to a neighbor is removed, when there is a tie the second shortest distance is considered and so on.

At the end of an iteration, the algorithm uses binary tournament to select the population of parents, performs the crossover and mutation, and returns to calculate the fitness of individuals while the stop criterion is not reached.

3.2.4 SPEA2+SDE

According to [59] Pareto-based MOEAs encounter difficulties confronting problems where the proportion of nondominated solutions is large especially when more than 3 objectives are used. Usually MOEAs try to balance convergence, i.e, the approximation to Pareto-front and diversity that is how well spread are solutions along the front. When the convergence criterion is undermined the leading role is played by the diversity criterion.

To deal with this problem instead of focusing on new dominance concepts that focus on convergences the SDE (shift-based density estimation) approach covers both the distribution and convergence of individuals. The SDE can be applied to any specific density estimator without the need for additional parameters. The pseudo-code in Algorithm 4 covers SPEA2+SDE:

Algorithm 4 SPEA2+SDE. Adapted from [87, 59]:

Input: $N, \bar{N}, g$

```

1: ...similar to lines 1-2 in SPEA2
2: for each individual in  $P_t$  and  $\bar{P}_t$  do
3:   Evaluate the fitness of the individual + SDE
4: end for
5:  $\bar{P}_{t+1} \leftarrow$  nondominated individuals in  $P_t$  and  $\bar{P}_t$ 
6: if  $|\bar{P}_{t+1}| > \bar{N}$  then
7:   reduce  $\bar{P}_{t+1}$  by means of the truncation operator + SDE
8: else if  $|\bar{P}_{t+1}| < \bar{N}$  then
9:   fill  $\bar{P}_{t+1}$  with nondominated solutions in  $P_t$  and  $\bar{P}_t$ 
10: end if
11: ...similar to lines 12-17 in SPEA2

```

The lines 3 and 7 of SPEA2+SDE are modified in relation to the original SPEA2 presented in the Algorithm 3. SDE is applied both in the fitness calculation and truncation operation because these two operations use density estimation.

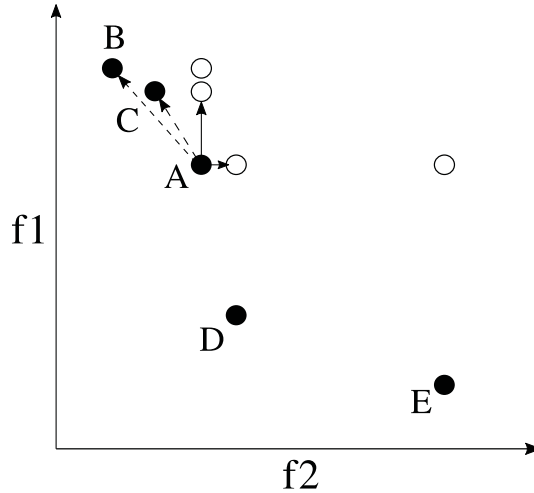


Figure 3.9: Illustration of SPEA2+SDE density estimation. Adapted from [59]

Density estimation for an individual is calculated in function of its position in relation to the position of other individuals. What SDE does while evaluating an individual is to move the others to the same position as the individual evaluated only for the objectives in which the other individuals have a better value as can be seen in Figure 3.9.

The measure used to estimate the density of the SPEA2 algorithm takes into account the Euclidean distance between A and its nearest k -th neighbors. It can be seen in Figure 3.9 that the two closest neighbors of A were C and B in the traditional SPEA2 density estimation and become D and C by using SDE.

What SDE seeks is to place individuals with low convergence in crowded regions so that they receive a high-density value and consequently are more easily eliminated during the evolutionary process.

3.2.5 MOEA/D

The MOEA/D (multiobjective evolutionary algorithm based on decomposition), according to [80], transforms the multi-objective problem of the Pareto frontier approximation into several mono-objective optimization sub-problems through functions responsible for the decomposition of the initial problem.

The pseudo-code of MOEA/D is presented in the Algorithm 5. The algorithm input consists of: the number of sub-problems, N ; a set of N weight vectors, here called W ; the number of weight vectors in the neighborhood of each weight vector, T ; and a stop criterion.

For each generation the algorithm maintains:

- the current population;
- for each solution, the fitness values corresponding to each sub-problem;
- the best values obtained for each objective;
- an external population composed of non-dominated solutions;

Algorithm 5 MOEA/D. Adapted from [80]

Input: N, W, T , stopping criterion

```

1: for  $i \leftarrow 1$  to  $N$  do
2:    $B(i) \leftarrow i_1, \dots, i_T$  where  $w^{i_1}, \dots, w^{i_T}$  are the  $T$  closest weight vectors to  $w^i$ 
3: end for
4: Initialize population  $P_0$ 
5: Initialize ideal and nadir objective vectors  $z^*, z^{nad}$ 
6: Set  $neval \leftarrow 0, iteration \leftarrow 0$ 
7: while Stopping criterion is not satisfied do
8:   for  $i \leftarrow 1$  to  $N$  do
9:     Randomly select two solutions  $x^{r_1}, x^{r_2}$  from  $P$ 
10:    Generate a candidate  $\bar{x}$  by applying crossover and mutation
11:    Evaluate the fitness value of  $\bar{x}$ 
12:    Update ideal and nadir objective vectors  $z^*, z^{nad}$ 
13:    Update neighboring solutions
14:     $neval++$ 
15:    Update the external population
16:   end for
17:    $iteration++$ 
18: end while

```

1. During the initialization procedure, the index containing the neighborhood of each weight vector is calculated by measuring the Euclidean distance, and the initial population is generated.
2. The reproduction procedure is done in two stages. Parents are selected from the sub-regions formed by the neighborhood of the weight vectors. If there is no neighboring solution in that region, then parents will be randomly selected from the entire population. There is also a small probability that the parents are selected from the entire population to increase exploration capacity, even if the sub-regions provide the necessary individuals. The mutation operators used by the MOEA/DD are SBX (simulated binary cross-over) and *polynomial mutation*.

3. If the generated solution performs better than a neighbor of the sub-problem considered, then that neighbor will be replaced by the new solution.
4. The external population is updated with the newly generated solution and the algorithm returns this population if the stop criterion is reached.

3.2.6 MOEA/DD

The MOEA/DD (multiobjective evolutionary algorithm based on dominance and decomposition), according to [57], proposes the combination of dominance and decomposition-based approaches to balance convergence and diversity.

As decomposition method, MOEA/DD originally uses the PBI approach presented in Equation 3.4, but any decomposition function can be chosen. A general framework of the MOEA/DD algorithm is presented in the algorithm 6:

Algorithm 6 MOEA/DD. Adapted from [57]:

Input: N , T , stopping criterion

- 1: Use Das and Dennis's method to generate weight vector W
- 2: ...similar to MOEA/D lines 1-5
- 3: Use the non-dominated sorting method to divide P into several non-domination levels $F_1, F_2, \dots, F_l$
- 4: Associate each member in P_0 with a unique subregion
- 5: **while** Stopping criterion is not satisfied **do**
- 6: **for** $i \leftarrow 1$ to N **do**
- 7: **if** $uniform(0,1) < \delta$ **then**
- 8: Randomly choose k indices from $E(i)$
- 9: **if** no solution in the selected subregions **then**
- 10: Randomly choose k solutions from P to form $\bar{P}$
- 11: **else**
- 12: Randomly choose k solutions from the selected subregions to form $\bar{P}$
- 13: **end if**
- 14: **else**
- 15: Randomly choose k solutions from P to form $\bar{P}$
- 16: **end if**
- 17: Generate a offspring population Q by applying crossover and mutation to $\bar{P}$
- 18: **for** each $x^c \in Q$ **do**
- 19: Execute the Update Procedure
- 20: **end for**
- 21: **end for**
- 22: **end while**

The operation of each procedure is described next. Initialization and reproduction procedures appear in Algorithm 6. The update procedure is highlighted in Algorithm 7.

Initialization procedure: the algorithm initialization has three aspects. In the first one, the initial population is obtained randomly from the decision space by a uniform distribution. The second aspect is the use of an adaptation of the method of Das and Dennis [17] (the same used in NSGA-III) to generate the weight vectors. In this adaptation, two layers of weight vectors are used. The objective of both layers is to overcome the problem of the computational cost necessary to treat objective spaces with many dimensions, which produce many weight vectors or few vectors, but with harm to diversity. The third

aspect is the calculation of the neighboring T for each weight vector using the Euclidean distance.

Reproduction procedure: Uses the same reproduction procedure as MOEA/D (see Section 3.2.5).

The Update procedure of MOEA/DD is equivalent to lines 11-15 in MOEA/D but more elaborated with different characteristics, for this reason, is presented in separate presented in the Algorithm 7:

Algorithm 7 MOEA/DD Update Procedure. Adapted from [57]:

```

1: Find the subregion associated with  $x^c$ 
2:  $P' \leftarrow P \cup \{x^c\}$ 
3: Update the non-domination level structure of  $P'$ 
4: if  $P'$  has only one level with non-dominated solutions then
5:   Remove the worst solution from the most crowded region
6: else
7:   if the last level  $F_l$  has only one solution then
8:     if the solution  $x'$  belongs to a region with more solutions then
9:       Remove  $x'$  from  $P'$ 
10:    else
11:      Identify the most crowded region and remove the worst solution from there
12:    end if
13:  else if the last level  $F_l$  has more than one solution then
14:    Identify the most crowded subregion associated with those solutions
15:    if more than one solution is associated to this region then
16:      remove the worst solution from this region
17:    else
18:      remove the worst solution from  $P'$ 
19:    end if
20:  end if
21: end if
22: update the non-domination level structure of  $P$ 

```

Update procedure: First, the region associated with the offspring solution is identified, and this solution is included in the parent population forming a hybrid population. Then we identify the levels of non-dominance of this population through an update calculation that avoids recalculating all levels with the inclusion of a new individual. When is necessary to remove an individual from the population there are two possibilities.

- If all members of the population are on the same level ($l = 1$), the worst solution in the most crowded sub-region is eliminated.
- If there is more than one level of non-dominance in the population, the last level begins, if it contains only one solution, then the density of the sub-region of that solution is evaluated. If there is more than one solution associated with the sub-region, then the individual from the level being verified is removed. If there is no other solution in the sub-region, this solution is maintained and the highest PBI solution is removed.

If the last level contains more than one solution, the most crowded region is identified. If more than one solution is associated with the more crowded sub-region, then the

worst solution (in terms of PBI value) is eliminated, otherwise, it indicates that each member of the last level is associated with an isolated sub-region, and the worst element of the population is eliminated (highest PBI).

3.2.7 MOEA/D-STM

The MOEA/D-STM [58] includes the use of an STM (stable matching model) as selection operator in a traditional MOEA/D algorithm.

The STM was initially proposed to solve the stable marriage problem [34], which consists of how to combine two sets of agents so that no unstable connections are made. Figure 3.10 illustrates an example of the stable marriage problem with couples:

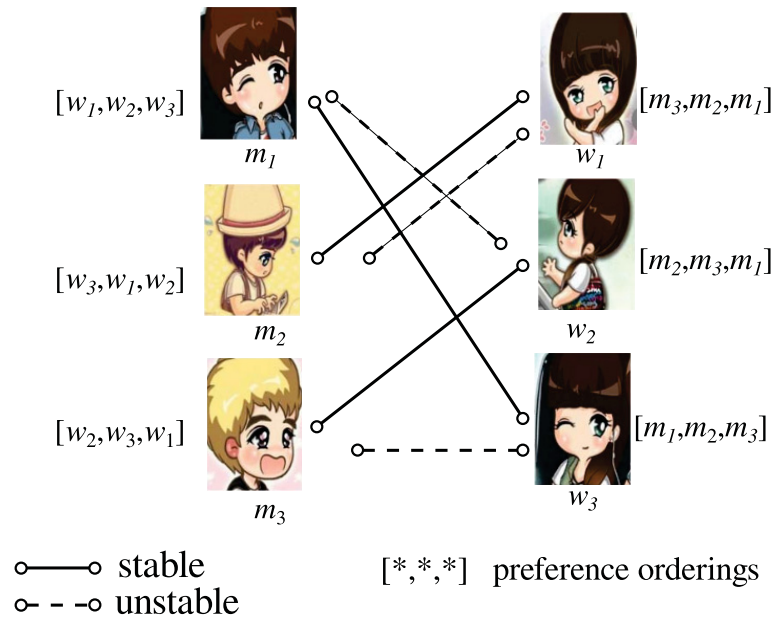


Figure 3.10: Example of Stable marriage problem. Adapted from [58]

In Figure 3.10 the matching $\{(m_1, w_2), (m_2, w_1), (m_3, w_3)\}$ is unstable because m_2 prefers w_3 to w_1 and w_3 prefers w_2 to w_3 .

In the case of the MOEA/D-STM, one group of agents is formed by the sub-problems and the other by the solutions. While the sub-problems want solutions with better values for the aggregation function, the solutions rank their preferences according to their distance to the direction vector of the sub-problem. The STM in MOEA/D-STM does the matching between solutions and sub-problems.

Next, we present an explanation of the algorithm focusing on the main differences to the traditional MOEA/D:

1. The initialization step is the same as MOEA/D.
2. Line 3 presents a change in relation to M, it defines the fraction of individuals of the population N selected for the next step, while in MOEA/D the next iteration occurs iterating through the entire population of size N .
3. In lines 5-9 the algorithm uses a neighborhood selection probability parameter to verify if the offspring will be obtained from the neighborhood or the entire population.

Algorithm 8 MOEA/D-STM adapted from [58]:

```

1: ... similar to MOEA/D initialization
2: while Stopping criterion is not satisfied do
3:   Let all indices of the subproblems whose objectives are MOP individual objectives  $f_i$  form the
   initial  $I$ . By using 10-tournament selection based on  $\pi^i$  select other  $\lfloor N/5 \rfloor - m$  indices and add them
   to  $I$ .
4:   for each  $i \in I$  do
5:     if  $uniform(0,1) < \delta$  then
6:        $E \leftarrow B(i)$ 
7:     else
8:        $E \leftarrow P$ 
9:     end if
10:    Randomly select three solutions  $x^{r_1}, x^{r_2}$ , and  $x^{r_3}$  from  $E$ 
11:    Generate a candidate  $\bar{x}$  by applying crossover and mutation,  $Q \leftarrow Q \cup \bar{x}$ 
12:    Evaluate the fitness value of  $\bar{x}$ 
13:    Update ideal and nadir objective vectors  $z^*, z^{nad}$ 
14:     $neval++$ 
15:  end for
16:   $R \leftarrow P \cup Q$ 
17:   $P \leftarrow STM(R, W, z^*, z^{nad})$ 
18:   $iteration++$ 
19:  if  $mod(iteration, 30) = 0$  then
20:    Update the utility of each sub-problem
21:  end if
22: end while
23: return  $P$ 

```

4. From lines 10 until 13 the offspring are generated by selecting the parents then applying crossover and mutation. The new solution is included to Q , the ideal and nadir vectors are updated and the number of algorithm evaluations increased.
5. After the population P and Q are merged into R the algorithm uses the STM to select a new population P in line 17.
6. Finally lines 19-21 MOEA/D-STM updates the utility function proposed in [81]. The utility of a sub-problem corresponds to how much improvement a new solution achieves in reducing the aggregation function values, i.e., the values of each mono-objective.

3.2.8 MOMBI-II

The MOMBI-II (many objective metaheuristic based on the R2 indicator), according to [44], presents an optimized version of the MOMBI algorithm (38), also based on the R2 indicator. Two aspects are primarily explored by MOMBI-II. The first aspect is the scalarization (aggregation) function, and the second is statistical information on the proximity of the population to the true Pareto front.

A scalarization (aggregation) function called ASF (achievement scalarizing function) [75] is incorporated by the MOMBI-II algorithm:

$$u_{asf}(\vec{v} : \vec{r}, \vec{\lambda}) = \max_{i \in \{1, \dots, m\}} \left\{ \frac{|v_i - r_i|}{\lambda_i} \right\} \quad (3.11)$$

where r is a reference point, w is a weight vector. The only difference between the Tchebycheff scalarization function 3.7 and the ASF function is the weight parameter.

The R2 ranking is performed according to the Algorithm 9:

Algorithm 9 R2 Ranking. Adapted from [44]:

```

1:  $p.rank \leftarrow p.\mu \leftarrow \infty$ 
2: for all  $\vec{w} \in W$  do
3:   for all  $p \in P$  do
4:      $p.\mu \leftarrow u_{asf}(p.\vec{f} : \vec{0}, \vec{w})$ 
5:   end for
6:   Sort  $P$  w.r.t.  $\mu$  and  $L_2$  in increasing order
7:    $rank \leftarrow 1$ 
8:   for all  $p \in P$  do
9:      $p.rank \leftarrow \min\{p.rank, rank\}$ 
10:     $rank \leftarrow rank + 1$ 
11:   end for
12: end for

```

Line 4 of Algorithm 9 shows that for each solution in the population is calculated the utility value that is how much the solution optimizes a weight vector. Then, in line 6, the rankings of P are calculated according to their utility value and the ties are solved using the Euclidean distance.

The main loop of the MOMBI-II is shown below:

Algorithm 10 MOMBI-II. Adapted from [44]:

```

Input: stopping criterion,  $W$ 
1: Initialize population  $P_0$ 
2: Evaluate population  $P_0$ 
3: Calculate the  $L_2$ -norm of objectives for  $P_0$ 
4:  $\vec{z}^{min} \leftarrow \vec{z}^*$  and  $\vec{z}^{max} \leftarrow \vec{z}^{nad}$ 
5: while stopping criterion is not met do
6:    $Q \leftarrow$  Perform parent selection by binary tournament in  $P_i$ 
7:    $P'_i \leftarrow$  applying crossover and mutation to  $Q$ 
8:   Evaluate population  $P_i$ 
9:   Calculate the  $L_2$ -norm of objectives for  $P_i$ 
10:  Normalize objective functions for  $P_i \cup P'_i$ 
11:  Execute R2 ranking algorithm ( $P_i \cup P'_i, W$ )
12:  Reduce population  $P_{i+1} \leftarrow \{P_i \cup P'_i\}$ 
13:  Update reference points ( $\vec{z}^{min}, \vec{z}^{max}, P_{i+1}, m$ )
14: end while
15: return  $P$ 

```

MOMBI-II requires a stopping criterion and a set of weight vectors W . During the main loop, the initial population is initialized and evaluated, and then the Euclidean distance is calculated between each individual of the initial population and the objective vectors. The minimum objective vector $\vec{z}^{min}$ and the maximum objective vector $\vec{z}^{max}$ receive respectively the ideal and nadir vectors obtained from the current population.

During the main loop until a stop criterion is reached:

- Selection of parent individuals through binary tournament and application of mutation and crossover operators to obtain an offspring population containing N individuals.

- The offspring population is evaluated, the Euclidean distance between each individual and the weight vectors is also calculated.
- The objective functions are normalized considering the union of the parent and offspring populations.
- The R2 ranking procedure is applied in the union of the parent and offspring populations and from this ranking the best individuals are selected to compose the new population that will be used in the next generation.
- Based on the new population, the reference points are updated.

The mechanism for updating reference points in MOMBI-II takes into account statistical information from previous generations, in light of the previous mechanism used by MOMBI to cause problems such as loss of diversity and population instability in final generations [44].

Initially, the ideal and nadir vectors of the current population are obtained and if there is an improvement in the vector $\vec{z}^{min}$ it will be updated, then $\vec{z}^{nad}$ is stored in a register and its vector of variances is estimated for each objective. If the maximum variance for all targets is high, then $\vec{z}^{max}$ will be updated using the nadir vector. Otherwise, each $\vec{z}^{max}$ component is examined, if the difference between its i -th value and $\vec{z}_i^{min}$ is less than the tolerance parameter ϵ , then it is marked and updated using the maximum target of $\vec{z}^{max}$. If the i -th value of the nadir vector is greater than the component, then $\vec{z}^{max}$ will be expanded and marked. Otherwise, if the variance is equal to zero and $\vec{z}^{max}$ has not yet been marked, then it will be marked and the average will be calculated between its previous value and its maximum value stored in the record. This tag lasts the same number of generations as the record is kept.

3.2.9 θ -DEA

The θ -DEA (θ -dominance-based evolutionary algorithm) algorithm, proposed in [79], implements non-dominated sorting replacing the Pareto dominance with θ -dominance.

As in NSGA-III and MOEA/D the θ -DEA uses reference points to ensure the diversity of solutions, however, it addresses the distribution of points through two layers obtained by the Equation 3.12. Using small values for H , which is the number of divisions considered in each objective axis, this approach avoids the problem of a very large number of reference points when considering a high number of dimensions d .

$$NPT = \binom{H_1 + d - 1}{d - 1} + \binom{H_2 + d - 1}{d - 2} \quad (3.12)$$

The distribution of the reference point layers can be seen in Figure 3.11:

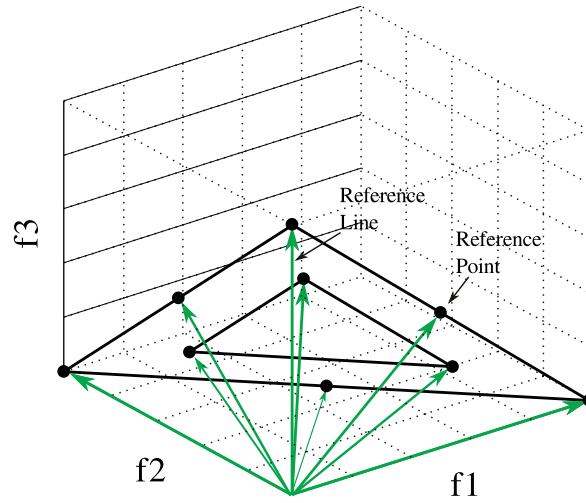


Figure 3.11: Two layer reference point structure. Adapted from (79)

The pseudo-code of the θ -DEA algorithm followed by its step-by-step description is presented below:

Algorithm 11 θ -DEA. Adapted from [79]:

```

1:  $\Lambda \leftarrow \text{GenerateReferencePoints}()$ 
2: Initialize Population  $P_0$ 
3: Initialize ideal and nadir objective vectors  $z^*, z^{nad}$ 
4: while stopping criterion is not met do
5:   Generate Child Population  $Q_t$ 
6:   begin
7:     Binary Tournament Selection
8:     Recombination and Mutation
9:   end
10:   $R_t \leftarrow P_t \cup Q_t$ 
11:   $S_t \leftarrow \text{Get Pareto non-dominated fronts } \{F_1, F_2, \dots, F_l\} \text{ of } R_t$ 
12:  Update  $z^*$ 
13:  Normalize  $S_t$ 
14:   $C \leftarrow \text{Apply Clustering operator in } S_t$ 
15:   $\{F'_1, F'_2, \dots\} \leftarrow \text{Apply } \theta\text{-Non-dominated-sort in } (S_t)$ 
16:   $P_{t+1} \leftarrow 0$ 
17:   $P_{t+1} \leftarrow \text{each non-domination level } \{F'_1, F'_2, \dots\} \text{ that fits entirely in } P_{t+1}$ 
18:   $P_{t+1} \leftarrow \text{complete with random selected individuals from the last partially fitted front.}$ 
19: end while

```

1. Initially the algorithm generates a set of N reference points and a random population with N members
2. The next step is to initialize the Ideal Point by selecting the lowest value found for the target f_i , and the nadir point by selecting the highest value for f_i found at P_0 .
3. During the main loop executed until the stop criterion is reached, the following steps are performed:
 - An offspring population Q_t is formed using recombination and mutation.

- P_t and Q_t are combined to produce a new population.
- S_t receives the elements of consecutive Pareto fronts $\{F_1, F_2, \dots, F_l\}$ until its size is equal to N or for the first time becomes greater than N . The last level which did not fit entirely in S_t will contribute with remaining necessary individuals.
- S_t is normalized and a clustering operator divides its members into N clusters.
- θ -dominance sorting is used to rank S_t at different levels $(F'_1, F'_2, \dots)$.
- The population P_{t+1} is filled with elements from the different levels starting with F'_1 . The elements of the last accepted level will be chosen randomly because θ -dominance combines convergence and diversity.

The θ -dominance is applied in the S_t population using a set of reference points. Each solution is associated with a cluster in the C set of clusters. Be $F_j(x) = d_{j,1}(x) + \theta d_{j,2}(x)$, $j \in \{1, 2, \dots, N\}$, where θ is a predefined penalty parameter, $d_{j,1}$ and $d_{j,2}$ are computed in the normalized goal space. The following concepts of θ -dominance can be established:

1. Given two solutions $x, y \in S_t$, it is said that x θ -dominates y , denoted by $x <_\theta y$, if $x \in C_j, y \in C_j$ and $F_j(x) < F_j(y)$, where $j \in \{1, 2, \dots, N\}$
2. A solution x is θ -optimal if there is no other solution x such as $x <_\theta x^*$.
3. All θ -optimal solutions in S_t form the θ -optimal set, the mapping of the θ -optimal set in the objectives space form the θ -optimal set forward.

Based on the listed definitions, the properties of irreflexivity, asymmetry, and transitivity can be established for θ -dominance.

- If a solution $x \in S_t$, then $x \not<_\theta x$.
- If two solutions $x, y \in S_t$ satisfy $x <_\theta y$, then $y \not<_\theta x$.
- If three solutions $x, y, z \in S_t$ satisfy $x <_\theta y$ and $y <_\theta z$, then $x <_\theta z$.

Given the above properties, an order at S_t can be established with the population being divided into different levels of θ -dominance.

3.2.10 IBEA

The IBEA (indicator-based evolutionary algorithm), proposed in [86], aims to use performance measures directly in the selection process of multi-objective optimization.

The pseudocode of IBEA is presented in Algorithm 12. As can be seen, the IBEA is a very simple algorithm, which iteratively removes the worst solution and updates the remaining individuals in the population. Then it performs the mating selection using binary tournament and updates the population for the next iteration. The main aspect of IBEA is its fitness assignment.

The fitness assignment is a ranking in the usefulness of individuals according to an optimization objective. As IBEA integrates an indicator in this process one possible solution is to calculate the loss in quality of the population if a specific solution is removed.

Algorithm 12 IBEA:

Input: N, g, κ

- 1: Initialize population P_0
- 2: **while** size of population P does not exceed N **do**
- 3: **for** each $x \in P$ **do**
- 4: evaluate fitness of x
- 5: **end for**
- 6: Remove from P the individual x with the smallest fitness value
- 7: Update the fitness values of remaining individuals
- 8: **if** stopping criteria is satisfied **then**
- 9: Consider individuals not dominated in P as result
- 10: **else**
- 11: $P' \leftarrow$ Perform binary tournament with replacement in P
- 12: Apply recombination and mutation to P'
- 13: $P \leftarrow P \cup P'$ and go to line 4
- 14: **end if**
- 15: **end while**

What IBEA uses in line 4 of Algorithm 12 to calculate the fitness is a scheme where the influence of dominating population members is amplified by a scaling factor, according to with Equation 3.13:

$$F(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} -e^{-I(\{x^2\}, \{x^1\})/\kappa} \quad (3.13)$$

Where the fitness of an individual is equal to the loss of quality if the individual x^1 is removed from the population. Small values of indicators I that preserve dominance, $I(\{x^1\}, \{x^2\}) < I(\{x^2\}, \{x^1\})$ if $x^1 < x^2$, will contribute much more to the overall fitness than larger values. The scaling factor κ needs to be greater than 0 and will depend on the indicator used and the problem.

3.2.11 HypE

The HypE (hypervolume estimation algorithm for multi-objective optimization) [5] was proposed as a fast alternative for the computational overhead drawback in other hypervolume-based MOEAs.

A hypervolume-based MOEA uses the value of the hypervolume indicator of the solutions to estimate their quality. This indicator evaluates the size of the area covered by a set of solutions, with the Pareto-optimal front having the maximum possible value [5].

There are many definitions for hypervolume, HypE uses the one presented in [5], basically calculating the area or volume of the subsets bounded by a front of solutions and a reference point (nadir point) using the Lebesgue measure [52] and applying Monte Carlo sampling [4]. The measure of hypervolume, in this case, can be exactly calculated for smaller datasets or approximated with a slight loss in accuracy for a large number of instances.

The Algorithm 13 presents a pseudo-code for HypE and its functioning is detailed in the following numbered steps.

Algorithm 13 HypE:

Input: stopping criterion, W

```

1: Initialize population  $P_0$ 
2: while stopping criterion is not satisfied do
3:   if  $m \leq 3$  then
4:      $\mathcal{F} \leftarrow$  Compute hypervolume of  $P$ 
5:   else
6:      $\mathcal{F} \leftarrow$  Estimate hypervolume of  $P$ 
7:   end if
8:   while  $|Q| < N$  do
9:     Randomly select two solutions  $a$  and  $b$  from  $\mathcal{F}$ 
10:    if  $v_a > v_b$  then
11:       $Q \leftarrow Q \cup a$ 
12:    else
13:       $Q \leftarrow Q \cup b$ 
14:    end if
15:  end while
16:   $P' \leftarrow$  applying crossover and mutation to  $Q$ 
17:   $R \leftarrow P \cup P'$ 
18:  Use the non-dominated sorting method to divide  $R$  into several non-domination levels  $F_1, F_2, \dots, F_l$ 
19:   $P \leftarrow \emptyset$ 
20:   $P \leftarrow$  each non-domination level that fits entirely in  $P$ 
21:   $k \leftarrow$  the number of elements remaining to complete  $P$ 
22:   $F_l \leftarrow$  the last front partially fitted in  $P$ 
23:  while  $|F_l| > k$  do
24:    if  $m \leq 3$  then
25:       $\mathcal{F} \leftarrow$  Compute hypervolume of  $F_l$ 
26:    else
27:       $\mathcal{F} \leftarrow$  Estimate hypervolume of  $F_l$ 
28:    end if
29:    Remove the solution with lower hypervolume from  $F_l$ , in case of tie select at random among
    tied solutions
30:  end while
31:   $P \leftarrow F_l$ 
32: end while

```

1. In line 2 the algorithm initializes its population by generating N solutions at random.
2. After starting the main loop in line 3, the algorithm proceeds with a mating selection, and according to m , the number of objectives, it computes or estimates the population hypervolume and assign a fitness value to each solution, lines 4-8.
3. After this, in lines 9-15, two solutions are selected from the current fitness assignment $\mathcal{F}$ and the one with higher hypervolume value is included in the mating population Q .
4. In line 17 crossover and mutation operators are applied in Q to generate an offspring population P' that will be merged with the previous population P into R in line 18.
5. The 20-31 indicates the environmental selection performed by the algorithm. First, the dominance levels are calculated for R and starting from the most dominant level the objects are moved to a new population. When the first level that cannot be transferred

completely is reached the algorithm successively calculates the fitness and removes the worst solution until the level size fits into P .

3.3 HYPERVOLUME INDICATOR

The hypervolume indicator [88, 89] is one of the most common ways to evaluate the set of solutions of MOEAs, being the only known measure that is sensitive to improvements, i.e., the approximation set that dominates will have a better hypervolume than the dominated one and also guarantees that a set with maximum possible hypervolume will have all nondominated objective-vectors (individuals in the population) [85].

In [85] we have a definition of hypervolume based on a simplified definition of attainment function. The set of all possible objective vectors is denoted as Ω , and the objective space is denoted as R^m , with m being the number of objective functions:

Attainment function: Given a set $A \in \Omega$. For $z \in R^m$. The attainment function α_A is given as.

$$\alpha_A(z) = \begin{cases} 1, & \text{if } A \geq \{z\} \\ 0, & \text{otherwise} \end{cases} \quad (3.14)$$

Hypervolume: the hypervolume I_A^* with reference point $(0, \dots, 0)$

$$I_A^* = \int_{(0, \dots, 0)}^{(1, \dots, 1)} \alpha_A(z) dz \quad (3.15)$$

The hypervolume is defined as the volume of the objective space enclosed by the attainment function and the axes. Figure 3.12 exemplifies the area covered in a two-dimensional space.

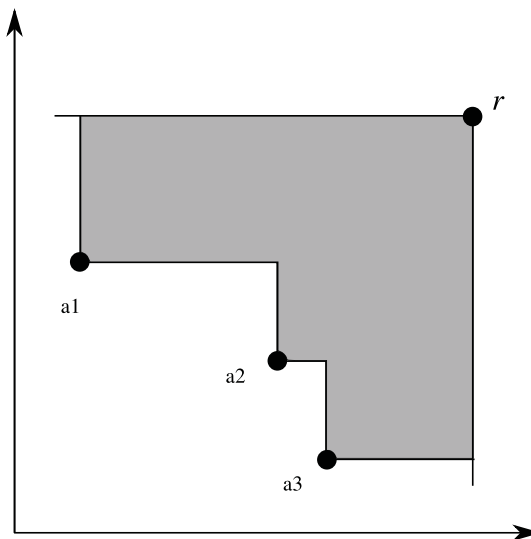


Figure 3.12: Illustration of hypervolume covering area. Adapted from [85]

In a two dimensional space, considering a minimization problem and r as a reference point (nadir point), the result set $A = a_1, a_2, a_3$ will dominate the gray area bounded by r , this region represents the hypervolume.

3.4 DISCUSSION

The set of MOEAs presented in this chapter is compiled in Table 3.1.

Table 3.1: Classification of studied MOEAs

Strategy	MOEA
Pareto-based	NSGA-II
Reference set	NSGAIII
Pareto-based	SPEA2
Diversity-based	SPEA2+SDE
Relaxed-dominance	θ -DEA
Aggregation	MOEA/DD
Aggregation	MOEA/D
Aggregation	MOEA/D-STM
Indicator	MOMBI-II
Indicator	HypE
Indicator	IBEA

The algorithms in Table 3.1 are an important component of the analysis that this work proposes. Six different categories of MOEAs are used.

The next chapter presents the related works in multiobjective clustering that, in addition to the cluster ensemble approach presented in Chapter 2.1, is another attempt to overcome the limitations of traditional clustering techniques.

4 RELATED WORKS

In this chapter, we first review multiobjective clustering related papers then we present works related to the data-driven analysis, and finally, we describe the three works more relevant to this study: MOCLE, MOCK and Δ -MOCK.

The term multi-objective clustering appears for the first time in [54] wherein a two-step process, initially a set of partitions is generated by multiple clustering algorithms and the second step integrates the partitions into a target partition using a conflict resolution technique to solve conflicts between clusters. Considering that the multiple objectives of this technique are the ones optimized by each clustering algorithm, there is no direct simultaneous optimization and the method proposed in [54] fits better as a cluster ensemble approach [26].

Many papers present clustering methods optimizing multiple objectives. In [67] one objective is based on the Euclidean distance, XB-index, and the other based on the point symmetry distance, Sym-index. In [62] the objectives to be minimized are the Jm-index and XB-index, both considering the compactness of clusters, however, the XB-index also considers the separation between clusters. In [3] the objectives are connectivity and cohesion.

With respect to the data-driven analysis the most closely related work to ours are [29, 30, 31, 2, 36]. In [29], the authors introduce MOCLE and compare its performance to those obtained by MOCK and the cluster ensemble method proposed in [68] on a collection of one synthetic and four real datasets. Their analysis showed that:

1. both MOCLE and MOCK presented comparable results in terms of their ability in recovering the underlying structure (true partition) in the datasets;
2. the sets of partitions suggested by MOCK's selection algorithm showed, for all datasets tested, a much poorer performance than those of MOCLE;
3. the size of the set of solutions produced by MOCLE, as expected, was much smaller than those of the complete set of MOCK.

The work in [30] also compares the performance of MOCLE and MOCK, but in the specific context of six cancer gene expression datasets (bioinformatics). The results obtained showed that, overall, MOCLE outperformed MOCK. The work in [31, 2], as compared to those in [29, 30], performed experiments with MOCLE and MOCK on a larger and more diverse collection of datasets. However, their analyses were focused not on the performance of MOCLE/MOCK, but on the model selection algorithms they proposed. Finally, in [36] the authors conduct a comprehensive data-driven investigation and comparison of Δ -MOCK with respect to MOCK. Their experimental results showed that the changes that they introduced in Δ -MOCK (see Section 4.2) impacted positively not only on computational efficiency but also on the clustering accuracy for complex datasets.

4.1 MOCK

MOCK (Multi-Objective Clustering with automatic K-determination) is a well-known algorithm for Pareto-based multi-objective clustering [41, 40]. MOCK uses an adaptation of the MOEA PESA-II (Pareto envelope-based selection algorithm version 2) [16], and optimizes the results of two objective functions, overall-deviation presented in the Equation 2.1, which estimates

the compactness criterion, and the connectivity, Equation 2.3, which estimates the degree of neighboring solutions belonging to the same cluster.

As objective functions, both *dev* and *con* are to be minimized [40]. Optimizing only *dev* leads to the increase of the number k of clusters in the partitions. Indeed, the minimum of such a function is attained as k is equal to the number n of objects in the dataset. In contrast, *con* tends to reach its minimum as the number of clusters decreases. In fact, depending on the value chosen for parameter L , different partitions could present the same optimal value for *con* [36].

MOCK uses a graph-based encoding called locus-based adjacency representation [40]: an individual $\mathbf{g}$ is composed of n genes ($g_1, g_2, \dots, g_n$), where n is the number of objects in the dataset to be clustered, and each g_i can take any value in $\{1, \dots, n\}$. For example, a value of j assigned to the i th gene means that objects i and j of the dataset must be in the same cluster.

An advantage of minimum spanning tree-based encoding is that while crossover and mutation operators have a detrimental effect in traditional clustering encoding, they do not affect as much a link-based encoding. MOCK uses a uniform crossover operator, randomly selecting which points from both parents will integrate the offspring, and uses a neighborhood-biased mutation where each object mutated is re-linked to one of the L nearest neighbors.

The initialization, as a way to provide a close initial approximation to the Pareto front, consists of two steps. First, generates half of the solutions by removing links from the MST (minimum spanning tree) according to a relevance criterion called *degree of interestingness* (DI). Intuitively, according to [42], a link between two objects i and j is considered *interesting* if neither of them is part of the other object's set of L nearest neighbors. In the second step, the other half of the population is generated by the k -means algorithm and converted to MST solutions. The MST-based representation is illustrated in Figure 4.1.

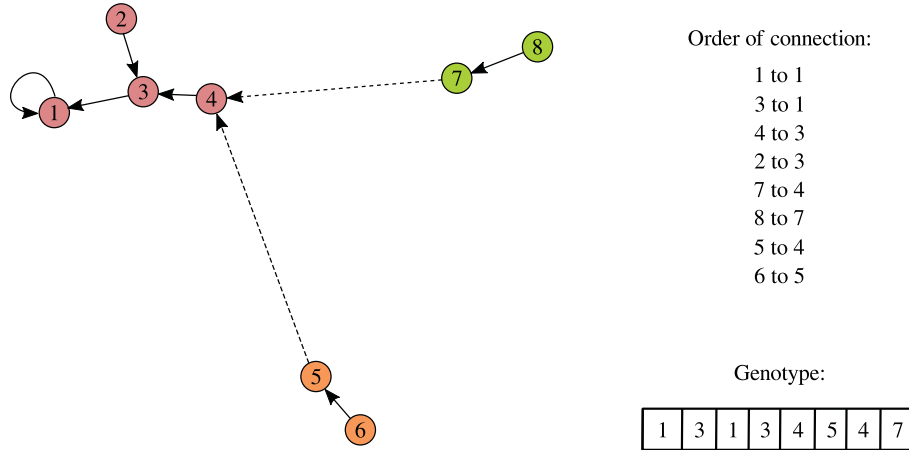


Figure 4.1: MST-based Representation. Adapted from [39]

The genotype indicated in Figure 4.1 represents the complete MST, to split the clusters represented by different colors it is necessary to replace the dashed links with a self-referential link such as in the object 1.

Finally, after the clustering phase, as previously mentioned, MOCK has a model selection phase. More specifically, the procedure used is based on the analysis of the shape of the trade-off curve obtained in the clustering phase (the Pareto front approximation) in comparison with trade-offs obtained for an appropriate null model, produced by clustering random data. Through this analysis, the algorithm provides an estimate of the partitions quality and determines a set of potential solutions (partitions) [40].

4.2 Δ -MOCK

The algorithm Δ -MOCK reviews the previous work, MOCK, making some modifications [36]. The main changes of the new version are in the encoding schemes and the initialization routine. In addition to these, the MOEA PESA-II [16] was replaced by NSGA-II [20]. According to [36], as NSGA-II is less elitist than PESA-II, it increases the method's ability to exploit the high-quality partitions generated during the initialization phase.

Regarding the encoding schemes, one of the main factors for MOCK's scalability was the length of the genotype being the same size N as the number of objects in the dataset, which can impact the convergence of the algorithm according to [36]. To deal with this problem Δ -MOCK has two possible encoding schemes inspired in the locus-based adjacency representation, Δ -locus and Δ -binary represented in Figure 4.2.

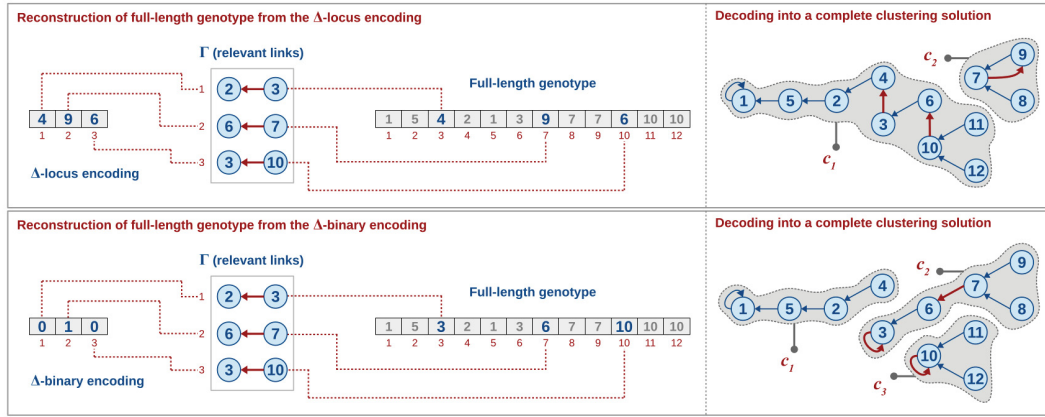


Figure 4.2: Δ -MOCK Representations. Adapted from [36]

More specifically, based on a variation of the measure *degree of interestingness*, all MST links are ranked into descending order. Then, based on a user-defined parameter, δ , they are classified either into the set of relevant links, Γ , or into the set of non-relevant, fixed links, Δ . The relevant links are those whose removal can split the MST into a set of connected components that could correspond to actual clustering structures. In the new encoding proposed only the links in Γ must be kept, that is, a $|\Gamma|$ -length genotype. More precisely, Γ contains the $|\Gamma| = \lceil ((100 - \delta)/100) * n \rceil$ links with largest DI. When δ is set to 0, the Δ -locus encoding is equivalent to the full-length locus-based adjacency representation. Indeed, δ has a direct impact not only on the length of the representation (the larger the value of δ , the smaller the length) but also on the maximum number k^{max} of clusters that can be encoded which is limited to $|\Gamma| + 1$. For example, for a dataset with 3100 objects, while MOCK's encoding can represent up to 3100 clusters, the Δ -locus encoding, setting δ to $\sqrt{n}$, is limited to ~ 56 clusters [36].

According to a relevance criterion a set, Δ , of non-relevant links is suppressed from the individuals and considered to be common fixed characteristics of all individual solutions while only the relevant links, in the set Γ , are optimized during the evolution process. Differences between the two encodings can be seen when applying the mutation operator:

Δ -locus: a mutation is equivalent to replacing an existing link in the minimum spanning tree with another link to one of the randomly selected nearest neighbors, i.e., Δ -locus can explore the replacement of the original MST links with other alternative links between objects in the dataset.

Δ -binary: as this encoding expresses the presence or absence of a link, then a mutation is equivalent to removing or inserting this link in the minimum spanning tree according to a probability, switching 0 and 1 only whether the original MST links are preserved (allele value 1) or removed (allele value 0).

According to [36], another factor that impacts MOCK's scalability is its two-phase initialization procedure (initial population), more specifically the second one in which partitions are created by removing all MST links crossing cluster boundaries of partitions produced by k -means (see Section 4.1). Motivated by this, they proposed an alternative procedure based on the original first phase employed by MOCK. More precisely, individuals are iteratively created by removing the most relevant links from the MST, that is, links belonging to Γ . In fact, to a great extent, there is a strong interaction between both the strategy employed to encode the solutions (the Δ -locus and Δ -binary schemes) and the strategy used to build the initial population.

Finally, in terms of objective functions, as MOCK, Δ -MOCK uses the connectivity (Equation 2.3), but instead of MOCK's overall deviation (Equation 2.1), it employs the intra-cluster variance as defined in Equation 2.2.

4.3 MOCLE

The clustering algorithm MOCLE (multi-objective cluster ensemble), proposed in [28, 29], combines characteristics from both cluster ensemble techniques [9] and multi-objective clustering methods [65]. The features of cluster ensemble techniques are present in the generation of MOCLE's initial population and, most importantly, in its crossover operator. For example, aiming at providing a "good" initial approximation to the Pareto front, as in strategies commonly used to generate base partitions in cluster ensembles, MOCLE's initial population (set of base partitions) is built by running several conceptually different mono-objective clustering algorithms (e.g., k -means and single-linkage clustering). Several parameter settings can also be used for each algorithm (e.g., different values for the number k of clusters).

The more different the algorithms used to generate the initial partitions, the greater the diversity of the population. Each individual is encoded as an array of sets $\{c_1^i, c_2^i, c_3^i\}$, and each set, c_1^i , corresponds to a cluster containing the objects and respective cluster labels.

Instead of the usual combination of all base partitions at the same time to generate a single consensus partition employed in cluster ensemble, in MOCLE, via a crossover operator which is itself a cluster ensemble technique, pairs of partitions are combined, iteratively, in a multi-objective optimization process. The optimization process is performed using the NSGA-II [20] algorithm as follows:

In line 2 of Algorithm 14, the base partitions obtained in the previous phase serve as the initial population of the NSGA-II algorithm.

Line 3 shows the two objective functions that the MOCLE algorithm tries to minimize, connectivity and overall deviation. After each new solution is created, these two functions are also calculated, lines 9 and 23.

Each offspring is generated, lines 5 and 19, using a binary tournament to select two solutions (partitions) and then determine a consensus partition from them using a cluster ensemble technique that will play the role of crossover operator. The MCLA and HBGF cluster ensemble techniques presented in the subsections 2.2.1 and 2.2.2 are the two crossover operators available in MOCLE. The absence of a mutation operator contributes to limit search space and return a more concise set of solutions.

The consensus partition generated at each iteration, depending on its quality concerning the values of the objective functions, can also be considered in the next combination. By doing

Algorithm 14 MOCLE+NSGA-II:

Input: $N, g, f_k(x)$

- 1: Population $P_0 \leftarrow$ set of partitions obtained with traditional clustering algorithms
- 2: Evaluate connectivity and overall-deviation of P_0
- 3: Assign Rank Based on Pareto dominance-sort to P_0
- 4: Generate Child Population Q_0
- 5: **begin**
- 6: Binary Tournament Selection of two partitions from P_0
- 7: Recombination using MCLA or HBGF ensemble
- 8: Evaluate connectivity and overall-deviation of Q_0
- 9: **end**
- 10: **for** $i=1$ to g **do**
- 11: $R_i = P_i \cup Q_i$
- 12: **for** each $x \in R_i$ **do**
- 13: Assign Rank Based on Pareto dominance - sort
- 14: **end for**
- 15: Generate sets of nondominated (F_i) vectors along PF known
- 16: Determine crowding distance between points on each front (F_i)
- 17: Select points on the lower front and outside a crowding distance to generate P_{i+1}
- 18: Generate Child Population Q_{i+1}
- 19: **begin**
- 20: Binary Tournament Selection of two partitions from P_i
- 21: Recombination using MCLA or HBGF ensemble
- 22: Evaluate connectivity and overall-deviation of Q_{i+1}
- 23: **end**
- 24: **end for**

so, the low-quality partitions are gradually discarded, while the best solutions are maintained for later combinations whether they are present in the base partition set as well as the generated ensembles. As a consequence of the multi-objective optimization process, differently of most cluster ensemble techniques that generate a single consensus partition as output, MOCLE yields a set of partitions that can be composed of consensus partitions, as well as base partitions, representing different trade-offs of the objective functions.

Finally, the core ideas of MOCLE are linked to specific objective functions and cluster ensemble techniques (crossover). In principle, a variety of existing objective functions and algorithms from cluster ensemble. For example, in terms of objective functions, like MOCK, MOCLE has been mainly implemented using *dev* and *con* [29, 30, 31, 2]. It has also been used, in the semi-supervised context, with the Gini index [27]. Concerning the crossover, the most used technique is the Meta-Clustering Algorithm (MCLA) proposed in [68]. The implementation relative to [26] also implements the Hybrid Bipartite Graph Formulation (HBGF) [32].

4.4 DISCUSSION

In the case of MOCLE, the special crossover operator based on cluster ensemble and the lack of mutation restrict the search space to the base partitions and their combination. This, together with the ability of each clustering algorithm (base partitions) in finding high-quality partitions according to its clustering criterion, gives MOCLE the ability to find a diverse set of solutions while keeping some level of concision of this set. Indeed, as reported in different work, MOCLE tends to produce a more concise set of solutions than MOCK [29, 30, 31].

A recurring concern in multi-objective clustering is how to get only one or a small number of partitions that represent the desired data structure, with multi-objective algorithms often returning a large set of solutions incomparable in terms of dominance. Works such as [67] and MOCK adopt heuristics to select only one or a reduced set of partitions from those returned during optimization. This type of approach is limited for example in MOCK this selection is made considering the distance between the solution and a control front and variations in either of them can affect the selected solution.

The strategy used by MOCLE to avoid a large set of solutions is to limit the search space by not using a mutation operator and seek to compensate for the lower degree of variation in the solutions with the crossover made by a cluster ensemble technique that by itself is a way of finding representative structures within the data set.

The next chapter presents the proposed approach to the study of different MOEAs applied in multi-objective clustering.

5 PROPOSED APPROACH

This chapter presents the jMocle (jMetal-based multiobjective cluster ensemble), the approach proposed in this research allowing to apply different evolutionary algorithms in the context of clustering through MOCLE's framework [26]. jMocle is available at <https://github.com/kultzak/jMocle>.

5.1 APPROACH DEFINITION

The core of this approach is the MOCLE algorithm, as defined in Section 4.3. With different choices in the set of clustering algorithms responsible for generating the initial population and especially in the evolutionary algorithms responsible for the optimization step. The overall structure is illustrated in Figure 5.1:

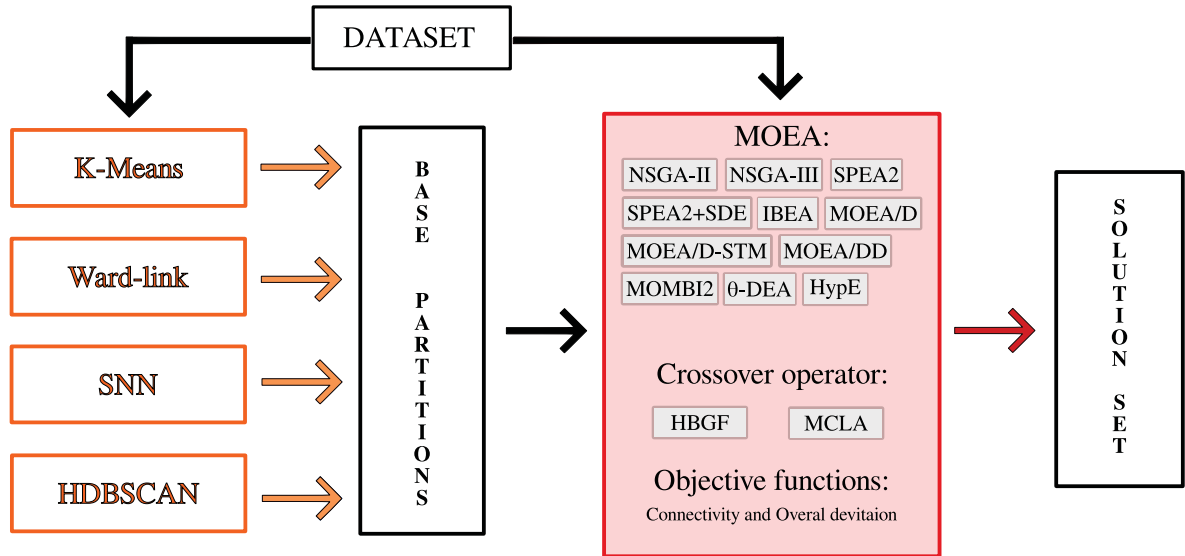


Figure 5.1: Overall structure of jMocle used to evaluate different MOEAs

The proposed clustering process starts with the set of traditional clustering algorithms that form the initial partitions. These partitions serve as input to the multiobjective optimization scheme composed by the selected(s) MOEA(s) and its crossover operator. In the end, the output is given as a set of solutions (partitions) that can be (1) preserved from the initial population or (2) new ones produced during the optimization via crossover operator (ensemble). The difference between this and MOCLE's approach relies just on the choice of alternative MOEA(s) and once this choice is made the whole framework performs as MOCLE. To make clear each MOEA will produce an individual result (set of partitions), not interacting or influencing amongst.

After a previous analysis of the clustering algorithms responsible for generating the initial partitions, it was decided to use the *k*-means, SNN, Ward-linkage, and HDBSCAN algorithms. Previous works with MOCLE have already used the *k*-means, Single-link, Complete-link and SNN [26], or *k*-means, Single-link, and spectral clustering [30]. We decided to keep *k*-means and SNN from [26] and selected Ward-linkage and HDBSCAN in a attempt to preserve the main characteristics of identification of clusters and improve the quality of initial partitions.

The k -means and Ward-linkage algorithms were obtained from the scikit-learn framework [66]. The SNN algorithm was used in an implementation developed by the authors [24]. The HDBSCAN algorithm was used from the implementation developed by [64].

As in MOCLE, the base partitions will serve as the initial population for an MOEA. Originally the MOEA used by MOCLE is the NSGA-II, in this work, it was varied applying the algorithms described in Section 3.2: NSGA-III, MOEA/D, MOEA/D-STM, MOEA/DD, SPEA2, SPEA2+SDE, IBEA, HypE, θ -DEA, and MOMBI2. The selection of these algorithms was made to provide great diversity in the way each algorithm approaches the multi-objective problem, also according to the availability of algorithms implementations.

The MOEAs implementation platform used was jMetal version 5.7, an object-oriented Java-based framework for multi-objective optimization [23]. The NSGA-II, SPEA2, MOMBI2, MOEA/D, MOEA/DD, MOEA/D-STM, and IBEA algorithms were used from implementations already present in jMetal. Additionally, algorithms were included from other sources available on the web: the NSGA-III was obtained from <https://github.com/gres-ufpr/nsga-iii>, SPEA2+SDE, HypE, and θ -DEA were obtained from <https://github.com/fritsche/hhco/tree/96185cdb545243d1914cfad06b532610f0055e78/src/main/java/br/ufpr/inf/cbio/hhco/algorithm>.

Regarding the ensemble methods used as a crossover operator, the HBGF was manually implemented based on the article by Fern and Brodley [32]. The MCLA was extracted from the implementation available in https://github.com/GGiecold/Cluster_Ensembles. Both algorithms work with graph partitioning as presented in Section 2.2, for this purpose the METIS [50] graph partitioning program was used.

The solution encoding is similar to the one used in MOCLE in which each individual is represented by a single partition, avoiding redundancy. In a representation based on a minimum spanning tree, such as the used in MOCK, a partition can be represented by more than one individual generating redundancy in the population. The representation used also avoids the need to correct individuals after the crossover to ensure that they are valid [26].

Table 5.1: Example data-set

pos	object	atribute x	atribute y
1	A	1	1
2	B	2	1
3	C	2	2
4	D	6	2
5	E	8	2
6	F	3	3
7	G	7	3
8	H	4	4
9	I	5	4
10	J	4	5
11	K	5	5
12	L	5	6

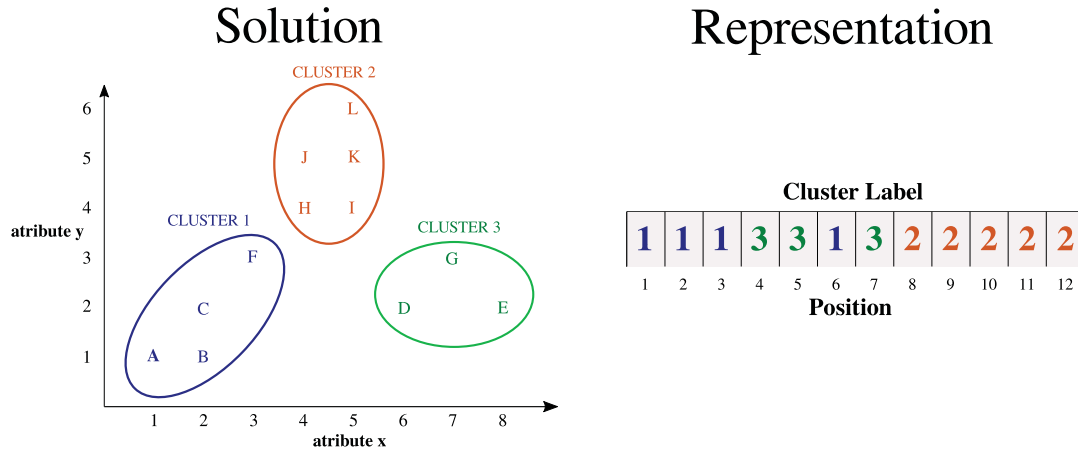


Figure 5.2: Solution encoding scheme.

Table 5.1 presents an example data set and one of the possible solutions is illustrated in Figure 5.2 together with the corresponding representation. jMocle encodes the solutions as an array with indices equivalent to the object's position in the data set and stores the respective cluster label generated for each object.

The objective functions implemented are the same as those used in MOCK and MOCLE and their use is justified based on the previous results of these studies and the balancing characteristic between both functions. While connectivity tends to converge to solutions with a single cluster, the global deviation tends to converge oppositely, with several clusters formed by single individuals [39].

5.2 DISCUSSION

This chapter presented jMocle a system for the evaluation of the different MOEAs applied in MOCLE to the clustering purpose. jMocle is built on the jMetal multi-objective optimization framework and uses MOCLE as its core.

In addition to the base partitions, for the optimization step, it is necessary to provide the same dataset that originated these partitions. Select the desired MOEA and the crossover operator and provide the necessary parameters that depend on the chosen MOEA. The data sets, initial population, and parameters used during the experiments will be presented in the following chapter along with the results obtained for the different MOEAs, and comparatives with MOCK and Δ -MOCK.

6 EVALUATION OF JMOCLE

The implementation of jMocle allows the investigation of some research questions, as follows:

RQ1: Exists a MOEA that is better for the clustering purpose considering MOCLE approach?

RQ1.1: Is there a MOEA capable of returning a more concise front?

RQ1.2: Is there a better MOEA to identify different structures or levels of refinement in a single dataset?

RQ2: Are the crossovers used in jMocle capable of contributing with the evolution the population according to the two objectives here considered?

RQ3: Is the initial population crucial to the performance of jMocle?

RQ3.1: Are other component changes capable of improving jMocle's performance?

RQ4: Are MOCK and Δ -MOCK better?

RQ5: How jMocle performs in the optimization compared to the traditional MOCK and a version of the MOCK initialized with the same population as jMocle?

This chapter describes the evaluation conducted to answer the different research questions and is organized as follows. Section 6.1 presents the data sets used in this work. Section 6.2 describes the methodology and parameters used to generate the base partitions. Section 6.3 presents the parameters used by the multi-objective algorithms. Finally, Section 6.5 presents and discusses the results and answers the research questions.

6.1 DATASETS

The data sets used in this work vary about the number of instances, number of dimensions, and number of clusters presented in the true partitions which are previously identified structures in the dataset. One dataset can have more than one true partition representing different refinement levels or different data conformations. A total of 19 datasets were used in the experiments 8 artificial and 11 real.

Table 6.1: Artificial Dataset Description. Number of objects n , number of attributes d , and number of clusters k^*

Dataset	n	d	k^*
ds2c2sc13_E1	588	2	2
ds2c2sc13_E2			5
ds2c2sc13_E3			13
ds3c3sc6-E1	905	2	3
ds3c3sc6-E2			6
spiralsquare	1500	2	6
D31	3100	2	31
tevc_20_60_1	4395	20	60

The dataset `ds2c2sc13` has three levels of refinement, E1, E2 and E3. The dataset `ds3c3sc6` has two levels. The refinement levels represent different structures of interest identified in the database as can be seen in Figures 6.1.

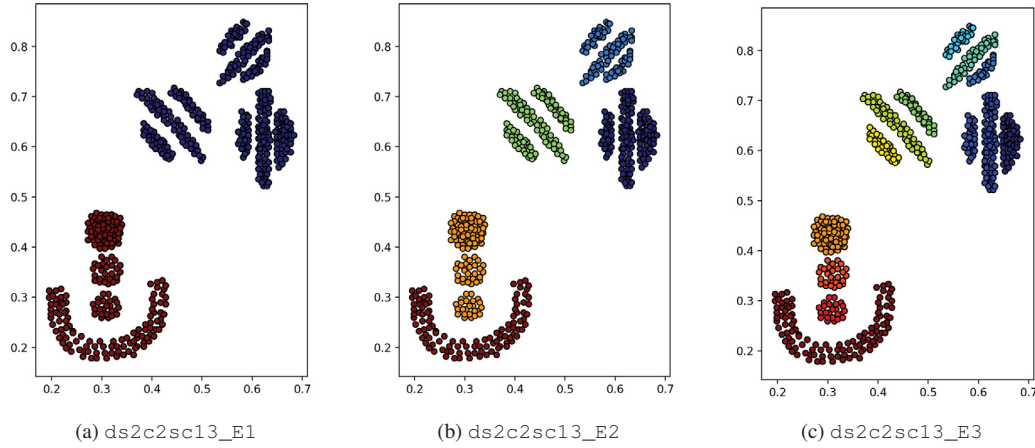


Figure 6.1: `ds2c2sc13` dataset structures.

These datasets were selected to make it possible to verify whether the multi-objective clustering technique is capable of returning different levels of refinement partitions in a single set of solutions, i.e., in a single set of solutions identifying different partitions equivalent to (a), (b) and (c) of Figure 6.1.

Table 6.2: Real Dataset Description. Number of objects n , number of attributes d , and number of clusters k^*

Dataset	n	d	k^*
Golub_E1	72	3571	2
Golub_E3			3
Seeds	210	7	3
Leukemia_E1	327	271	3
Leukemia_E2			7
Libras	360	90	15
OptDigits	5620	62	10
Frogs_MFCC_E1	7195	22	4
Frogs_MFCC_E2			8
Frogs_MFCC_E3			10
UKC1	29463	2	11

The dimensions of the dataset Golub consist of genes found in tumor samples, each instance corresponds to a sample. The Seeds dataset stores information about structures in different varieties of wheat. The Leukemia dataset consists of gene expressions from patients with different classifications of leukemia. The Libras dataset is composed of representations of different hand movements of the Brazilian Sign Language (LIBRAS). The OptDigits dataset stores data on handwritten digits. The Frogs_MFCC dataset stores data from different species of anurans according to their calls. Finally, the dataset UKC1 stores data about street-level crime in the U.K. [22].

Among the real datasets, there are also examples with multiple versions of identified structures. This is the case of datasets Golub, Leukemia, and Frogs_MFCC.

6.2 GENERATION OF THE BASE PARTITION SET

The methodology and parameter settings of the algorithms used to generate the initial population were as follows:

***k*-means:** The algorithm depends on an input value, K , corresponding to the number of clusters in which the database will be partitioned. Partitions were generated between a minimum value, K_{min} , and a maximum value, K_{max} . For all databases were defined the values $K_{min} = 2$ and $K_{max} = 2 * \text{the number of clusters in the known structure (true partition)}$. *k*-means is not a deterministic algorithm because it depends on the initialization of the centroids. Thus, it was performed 30 times for each K value with random centroids and returned for each K the partition with the lower inertia, e.g., sum of squared distances to the cluster centroids.

Ward-linkage: This algorithm depends equally on a K parameter that was varied in the same way as the *k*-means but is deterministic so only one execution per K was necessary.

SNN: Several parameters were defined for this algorithm and some varied according to a specified range. The number K of clusters in the partition cannot be defined, as well as the number of results that the algorithm returns.

- *nn*: 10, 20, 30, 40, 50 if dataset < 1500 instances, else: 20, 40, 60, 80, 100. For some cases where the SNN did not return any partitions, the following configuration was used. 70, 90, 110, 130, 150 if dataset < 1500 instances, else: 120, 140, 160, 180, 200.
- *topic* and *merge*: 0, 0,2, 0,4, 0,6, 0,8 and 1.
- *strong*: 0,25.
- *noise* and *label*: 0. The *nn* parameter was defined according to series of individual tests performed in the datasets, the other parameters were chosen according with [26].

HDBSCAN: The *min_cluster_size* parameter was varied from 5 in 5 to 100, from 100 in 100 to 1000, then 2000, 3000, 4000, 5000. This parameter was varied until the maximum number of instances of each base, therefore, a dataset with 1500 instances will have a variation between 5 and 1000. After exhaustive tests to parameter definition, since the HDBSCAN algorithm tends to produce a distinct number of solutions that is too low depending on the dataset, for this reason there are such big ranges in the *min_cluster_size* parameter. It was assigned 1 to the parameter *min_samples* that indicates how conservative the clustering is, this minimum value indicates that the algorithm will be the least conservative possible allowing points that would be considered noise to be part of a cluster.

As in the algorithms *k*-means and Ward-linkage it is possible to choose the number of clusters of the solution and the variation of the parameter K is the same for both, so the number of solutions returned by them is always twice the number of clusters of the true partition $k^* - 1$, because the variation starts at 2. These algorithms always contribute with the same number of partitions in the initial population.

It is not possible to choose the number of clusters in the output of the SNN and HDBSCAN algorithms. It may be that for some parameter settings these algorithms return the same partition or partitions outside the desired range for the number of clusters.

To avoid repeated or out-of-range partitions in the initial population, filters have been applied that removes this type of occurrence. One consequence of this is that SNN and HDBSCAN generally contribute with fewer solutions to the initial population than k -means and Ward-linkage.

The HDBSCAN algorithm can invariably return objects classified as noise by assigning the label -1. In these cases, all objects considered noise in a partition were grouped into a new cluster. By doing this the partitions generated by the HDBSCAN could be accepted since for example the representation and crossover used in this work cannot handle noise.

6.3 MULTIOBJECTIVE CLUSTERING ALGORITHMS PARAMETERS

Here are defined the parameters used in jMocle, MOCK, and Δ -MOCK for the tests.

6.3.1 jMocle

First are defined the parameters and operators used in common by all MOEAs tested in jMocle:

Population size: the parameter N corresponds to the size of the regular population maintained in each algorithm. It is based on this population that the successive iterations occur. In this work N has the size of the initial population, i.e., it is equivalent to the number of partitions generated by the clustering algorithms that formed the base partitions. In cases where there is an odd number of partitions in the initial population, then N is rounded to an even number because some algorithms do not accept odd populations due to the selection operator, the repeated element is the last one defined by alphabetical order of the solution labels. Binary tournament selection of two parents in an even population would result in a single solution being left off, since jMetal framework does not allow, and jMetal's implementation do not allow this.

Selection operator: all algorithms use a binary tournament to select the solutions that will be recombined. In this method, two solutions are chosen randomly in a population and the best of them is returned.

Crossover operator: HBGF or MCLA, presented in Sections 2.2.1 and 2.2.2, are the cluster ensemble techniques used as crossover in this work and previously used in MOCLE [26, 30].

Number of parents and offspring: the proportion of two parents to one offspring (ensemble of two partitions) is common to all MOEAS applied in this work. this approach was selected to follow the implementation of [26] which used two partitions for each ensemble.

L parameter: this parameter corresponds to the number of neighbors considered during the calculation of the connectivity objective function. The specified value is 10 as stipulated in [40, 36].

Number of evaluations: considering the value used by the original MOCLE [26], and some preliminary experiments also carried out by this study, the number of iterations was set at 50, given more iterations not constituting significant differences in the composition of the results.

Next are the parameters that only some MOEAs use according to the strategy adopted by the algorithm:

Neighborhood size: the algorithms MOEA/D, MOEA/DD and MOEA/D-STM have a neighborhood size parameter. After previous tests, the value was defined as 2.

weight vectors: the set of weight vectors used by some MOEAs such as NSGA-III and MOEA/D was always obtained through uniform distribution with the number of vectors equals N .

aggregation function: the aggregation function used by MOEA/D, MOEA/DD and MOEA/D-STM is the weighted sum. For a given solution, we sum up the values of each objective multiplied by a weighting coefficient. The sum of the weighting coefficients for any solution must be equal to 1, transforming that way the two objectives into a single objective function.

6.3.2 MOCK and Δ -MOCK

To execute MOCK and Δ -MOCK, we employed the same general settings as reported in [40, 36]:

Population size: both MOCK and Δ -MOCK use an internal population of size 10 and an external population of size 1000, with an initial population of size 100 in MOCK and in Δ -MOCK size $2k$, with k being an user input parameter, for this work it was considered a k of $2k^*$, so it varies according to number of clusters in the true partition of each dataset.

Number of generations: 500 for MOCK and 100 for Δ -MOCK, according to its respective implementation default values.

Distance function: Euclidean distance.

L parameter: This is the same parameter used in the definition of connectivity in jMocle but besides this function for MOCK and Δ -MOCK the L parameter is involved with initialization and mutation.

k_{max} of initial population: the maximum number of clusters in the initial population was set to $2k^*$, where k^* is the number of clusters in the true partition representing the dataset.

representation: this is a Δ -MOCK exclusive parameter, in this work we used Δ -locus scheme for, as pointed out in [36], it tends to produce better results than the Δ -binary one.

length of Δ -locus scheme: was defined as a function of $\sim 5\sqrt{n}$, where n is the number of objects in the dataset. This is one of the heuristics employed in [36] to chose the value of δ . According to the authors, in most of the cases they analyzed, the setting of δ did not seem to affect the performance in terms of clustering recovery.

To execute MOCK, we used the software available at <https://personalpages.manchester.ac.uk/staff/Julia.Handl/mock.html>. The implementation of Δ -MOCK is available at <https://github.com/garzafabre/Delta-MOCK>.

6.4 PERFORMANCE ASSESSMENT

This work uses two main indicators, we used hypervolume presented in Section 3.2.11 to evaluate the performance of the different MOEAs among themselves. The indicator was obtained from the PISA library [8], which contains modules with multi-objective optimizers, benchmark problems,

and modules for performance assessment. The hypervolume is among the most used indicators to evaluate the fronts produced by multi-objective algorithms, besides being monotonically related to the dominance exerted by the fronts, the more dominant the greater the hypervolume [5].

The ARI (adjusted Rand index) presented in Section 2.3 is another indicator we used because it allows comparing the solutions obtained in this study with previously determined known structures (true partitions). Since all datasets selected for this work have true partitions it is possible to evaluate the quality of new solutions using ARI.

To calculate Hypervolume and ARI, it was considered the results of 30 executions of each MOEA. Some results were selected to answer the research questions ahead, to reduce the number of tables and information presented in this study and limit its scope. The proposed system, some raw results, dataset information and complementary material are available at <https://github.com/kultzak/jMocle> and <https://github.com/kultzak/multiobjective-clustering-datadriven-analysis>

6.5 RESULTS AND ANALYSIS

The research questions will be answered below according to the experiments performed:

RQ1: Exists a MOEA that is better for the clustering purpose considering MOCLE approach?

To answer this question, we first performed Friedman’s test [33], comparing the results of the hypervolume indicator of all algorithms for all databases using the HBGF crossover. This test indicates if there is a significant difference between the results of each MOEA concerning the generated fronts. We used a significance level of 5%, i.e., there is a 5% risk that a difference indicated as significant is not significant.

Table 6.3: Average Rankings and standard deviation obtained in the Friedman test of MOEAs hypervolume results across the 19 datasets using HBGF crossover

Algorithm	Ranking
MOEADD	4.053±2.070
MOMBI2	5.289±2.551
NSGAII	1.842±0.889
HypE	9.632±2.411
MOEAD	9.079±1.489
NSGAIII	7.158±2.059
SPEA2SDE	4.026±1.690
ThetaDEA	6.526±2.980
IBEA	6.895±2.511
SPEA2	3.684±1.786
MOEADSTM	7.816±1.914

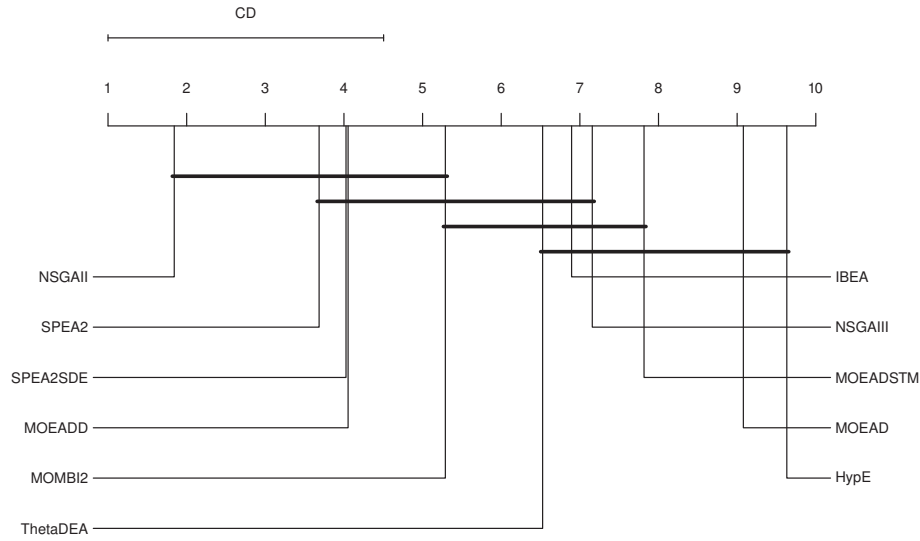


Figure 6.2: Critical Difference in the hypervolume using HBGF crossover.

The horizontal lines in Figure 6.2 connect statistically equivalent MOEAs. This test did not return an individual leading algorithm. The first five algorithms presented equivalent results and according to the MOEAs classification presented in Section 3.2, we have two Pareto-based, one Relaxed-dominance-based, one Aggregation based that also considers dominance, and one indicator-based MOEA. There is a higher presence of algorithms driven by dominance in the first level.

Next, the solutions generated by each algorithm were also evaluated according to the ARI, to identify which ones return the best partitions, applying the Friedman test to this index value.

Table 6.4: Average Rankings and standard deviation obtained in the Friedman test of MOEAs ARI results across the 19 datasets using HBGF crossover

Algorithm	Ranking
MOEADD	4.684±2.237
MOMBI2	5.684±1.879
NSGAII	5.947±1.776
HypE	8.447±3.228
MOEAD	6.368±3.256
NSGAIII	5.421±2.066
SPEA2SDE	5.842±2.213
ThetaDEA	5.658±2.724
IBEA	5.526±2.741
SPEA2	4.947±2.139
MOEADSTM	7.474±2.598

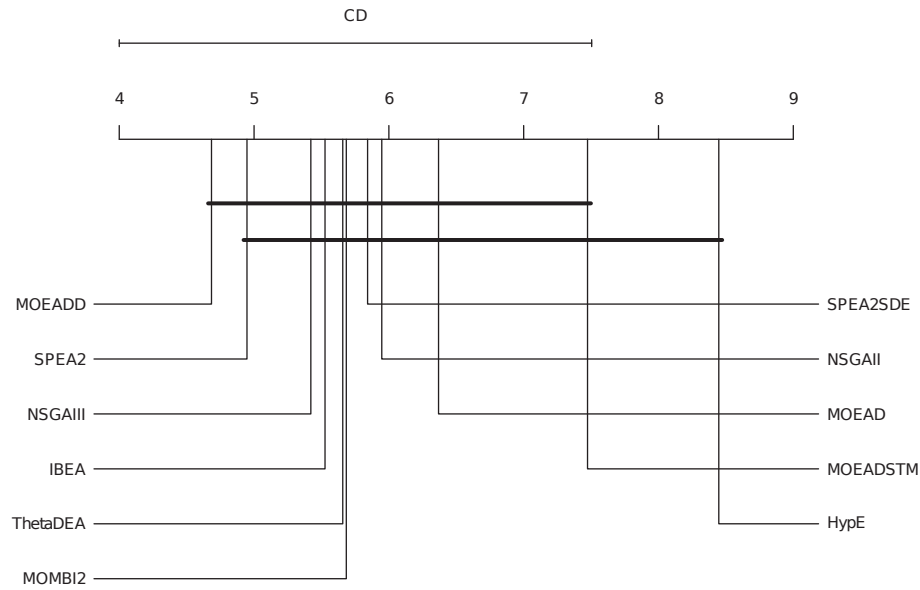


Figure 6.3: Critical Difference in the Adjusted Rand Index.

Table 6.4 and Figure 6.3 indicate that based on the ARI there is no significant difference between the algorithms. An exception occurs only for values returned by MOEA/DD and HypE when compared to each other.

Next, the MOEAs were executed using the MCLA crossover in the twelve following data sets: ds2c2sc13_E1, ds2c2sc13_E2, ds2c2sc13_E3, Frogs_MFCC_E1, Frogs_MFCC_E2, Frogs_MFCC_E3, Golub_E1, Golub_E3, Leukemia_E1, Leukemia_E2, Libras, OptDigits, and spiralsquare. Not all datasets were used with MCLA because in some previous tests performed, the METIS tool responsible for partitioning graphs presented instability for some bases that were left out of these tests. After an investigation of the problem, it was discovered that the instability was related to this question <http://glaros.dtc.umn.edu/gkhome/node/568> available in the METIS distributor forum with the same message being exhibited and in some cases an interruption of METIS functioning, but no further resolution was found so the datasets where the problem occurred were not used because not all the 30 runs could be completed in viable time in most cases.

Table 6.5: Average Rankings and standard deviation obtained in the Friedman test of MOEAs hypervolume results across the 12 datasets using MCLA crossover

Algorithm	Ranking
MOEADD	5.750±2.126
MOMBI2	5.958±3.058
NSGAII	4.833±1.724
HypE	10.917±0.276
MOEAD	9.375±1.244
NSGAIII	5.083±2.532
SPEA2SDE	3.167±1.951
ThetaDEA	4.333±1.312
IBEA	6.958±1.561
SPEA2	1.583±1.115
MOEADSTM	8.042±1.506

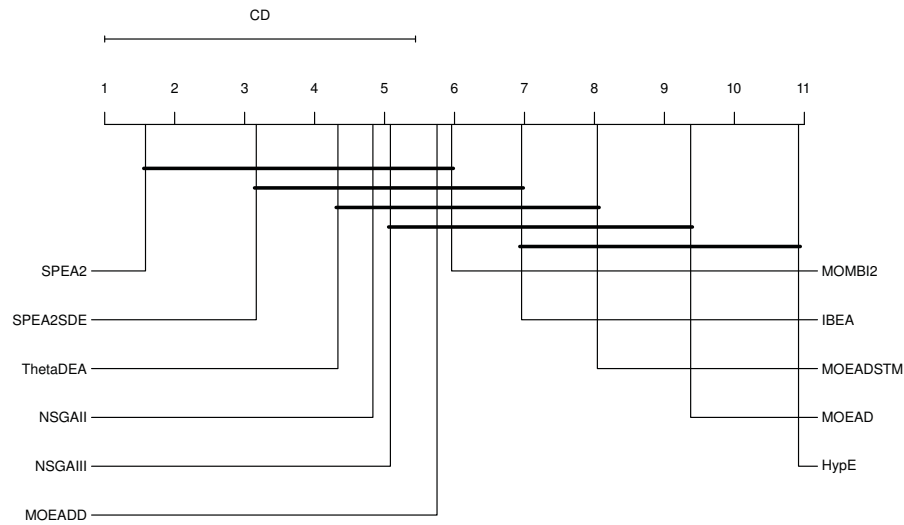


Figure 6.4: Critical Difference in the hypervolume using MCLA crossover.

The critical difference in the hypervolume indicator with the MCLA crossover presented five horizontal lines, indicating five levels of difference between the algorithms. There was a greater difference than the HBGF test, however, the first seven MOEAs did not present significant differences. NSGA-II, SPEA2, SPEA2+SDE, MOEA/DD, and MOMBI2 MOEAs appear in the first level of significance using either HBGF or MCLA crossover.

Table 6.6: Average Rankings and standard deviation obtained in the Friedman test of MOEAs ARI results across the 12 datasets using MCLA crossover

Algorithm	Ranking
MOEADD	5.500±2.930
MOMBI2	5.500±3.452
NSGAII	5.625±2.311
HypE	8.750±3.003
MOEAD	8.667±2.528
NSGAIII	5.625±1.927
SPEA2SDE	3.875±2.022
ThetaDEA	4.750±2.554
IBEA	6.667±2.786
SPEA2	3.542±1.797
MOEADSTM	7.500±2.574

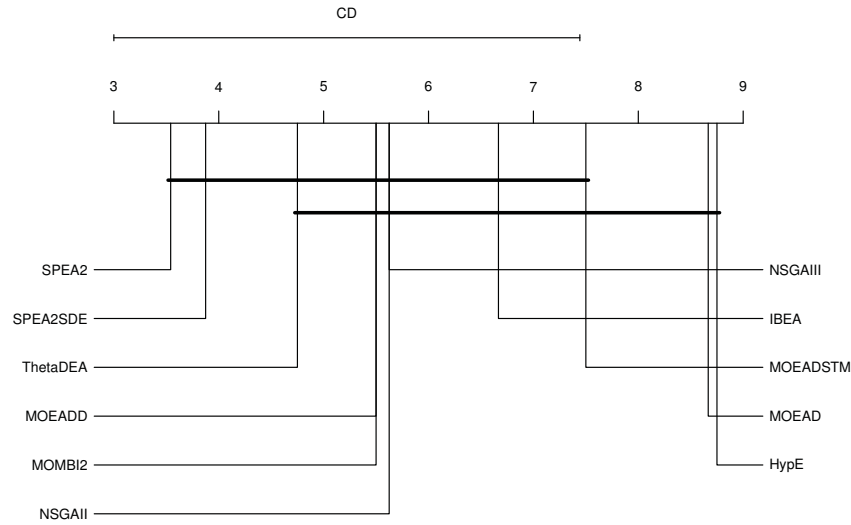


Figure 6.5: Critical Difference in the ARI using MCLA crossover.

When comparing the ranking obtained by Friedman tests in the ARI of the solutions obtained by the MOEAs while using the MCLA crossover in Table 6.6 and Figure 6.5, there was no evidence to point that the first nine algorithms were statistically different. From the third to the eleventh algorithm there is also no evidence of significant difference.

There are different levels of significance when comparing the Friedman tests in the hypervolume results but it is verified that there is no direct equivalence of these results in terms of ARI. The Friedman tests in the ARI results presented a flatter distribution. The results presented indicate that no MOEA was better alone and that especially when ARI was analyzed the behavior of all algorithms was statistically similar considering the set of databases of this study.

RQ1.1: Is there a MOEA that returns a more concise front?

Another way to evaluate MOEAs in jMocle is to compare the number of solutions returned by each of them. Considering the clustering problem, the fewer the number of solutions returned, the better, as long as they do not lose quality. As presented in Chapter 4 some multiobjective clustering algorithms use heuristics to select the best partitions among the returned set that is sometimes too large for analysis. A smaller number of solutions returned by the algorithm facilitates decision making when an expert needs to choose a partition.

Table 6.7: Mean and standard deviation for the number of solutions produced in 30 jMocle+HBGF runs. In bold are the initial population size

PROBLEM	HypE	MOEA/D	MOEA/D-STM	NSGAII	SPEA2	θ -DEA	IBEA	MOEA/DD	MOMB12	NSGAIII	SPEA2-SDE
D31 / 168	40±21.4	54±5.3	43±3.5	65±0.2	65±1.1	59±1.1	58±6.9	65±0.4	58±2.6	60±5.1	66±3.3
ds2c2sc13_E1 / 10	1±0.0	1±0.2	1±0.2	1±0.7	1±1.3	1±0.6	1±0.2	1±0.4	1±0.0	1±0.0	1±0.0
ds2c2sc13_E2 / 30	8±0.0	7±0.3	8±0.2	8±0.0	8±0.0	7±0.7	8±0.2	8±0.0	8±0.0	8±0.2	8±0.2
ds2c2sc13_E3 / 77	26±0.5	24±3.1	24±1.4	26±0.0	26±0.2	24±0.7	26±1.1	26±0.2	26±0.4	26±0.8	26±0.2
ds3c3sc6_E1 / 19	4±2.0	9±1.2	9±1.1	10±0.0	10±0.4	10±0.7	8±0.6	10±0.6	8±0.9	9±0.4	10±0.5
ds3c3sc6_E2 / 37	5±3.4	15±1.5	12±1.9	17±0.2	17±0.3	17±1.1	15±1.2	17±0.8	17±0.3	13±0.2	17±0.6
Frogs_MFCC_E1 / 20	6±1.7	7±1.2	8±1.0	8±0.2	8±1.1	7±0.5	8±1.4	8±0.5	8±0.2	7±0.2	8±0.8
Frogs_MFCC_E2 / 41	10±3.7	14±1.7	15±0.9	15±0.0	17±3.5	14±0.4	14±1.4	15±0.0	15±0.8	14±0.6	17±3.9
Frogs_MFCC_E3 / 51	12±4.9	17±2.2	17±1.7	19±0.0	24±5.4	19±0.4	17±3.6	19±0.4	19±1.0	19±1.4	21±3.3
Golub_E1 / 11	4±1.9	5±1.0	6±0.5	8±1.0	9±1.6	6±0.7	7±0.7	6±0.6	6±0.7	6±0.3	8±1.2
Golub_E3 / 17	4±1.9	7±1.1	8±0.6	10±1.0	10±1.4	9±0.8	9±1.0	9±1.0	9±0.7	8±0.5	10±1.1
Leukemia_E1 / 17	5±0.4	5±0.2	5±0.0	5±0.0	5±0.2	4±0.3	5±0.0	5±0.0	5±0.0	5±0.0	5±0.2
Leukemia_E2 / 39	14±0.3	13±0.6	14±0.7	14±0.0	14±0.2	12±0.4	14±0.0	14±0.7	14±0.0	14±0.0	14±0.0
Libras / 88	21±7.4	31±3.0	30±2.9	36±0.2	38±0.9	35±1.4	31±2.9	37±1.7	36±2.0	34±1.0	38±1.7
OptDigits / 57	19±5.9	21±3.8	19±1.8	22±0.2	22±0.5	17±1.2	22±1.5	22±0.4	17±1.5	21±2.1	22±0.9
spiralsquare / 36	7±4.3	13±2.4	7±1.4	15±0.8	15±0.9	11±0.5	11±1.7	14±0.5	12±1.2	8±0.4	15±0.8
Seeds / 17	3±1.4	7±1.0	8±0.5	9±0.2	10±0.7	8±0.7	9±1.0	9±0.5	9±0.7	8±0.4	10±0.7
tevc_20_60_1 / 274	66±0.2	62±6.2	64±6.1	66±0.0	66±0.2	66±0.0	65±3.6	66±0.2	66±0.0	65±2.9	66±0.2
UKC1 / 56	8±4.9	13±1.8	13±1.3	14±0.0	14±0.0	12±0.7	13±1.6	14±0.0	14±0.2	9±0.2	14±0.0

Table 6.7 presents the average sizes of the solution sets returned by each algorithm for each dataset, which can be compared with the total size of the initial population in bold. We notice that in most cases there is a small difference in the average number of solutions returned by each algorithm depending on the dataset considered. In this experiment it was identified that the HypE algorithm produced more concise sets of solutions, but when analyzing the results of RQ1 it is observed that this MOEA was always in the last positions indicating that this conciseness of the results is possibly being harmful, it may be missing out on potential good partitions in the set of solutions.

A general aspect of the technique that can be observed is the smaller number of solutions compared to the initial population. The dataset `tevec_20_60_1` that has 274 individuals in the initial population returns in most MOEAs an average of 66 solutions.

RQ1.2: Is there a better MOEA to identify different structures or levels of refinement in a single dataset?

For this experiment, we used datasets with more than one version. We verified the possibility that when executing the version of the dataset with the highest number of clusters the MOEAs were able to return among the resulting partitions the other data conformations with fewer clusters for the same dataset.

Table 6.8: Identification of different data structures in a single dataset using HBGF crossover. ARI, number of clusters between parenthesis.

PROBLEM	subproblem	HypE	MOEA/D	MOEADSTM	NSGAII	SPEA2	θ -DEA	IBEA	MOEA/DD	MOMBI2	NSGA-III	SPEA2+SDE
ds2c2sc13_E3	ds2c2sc13_E1	0.3809 (5)	0.3809 (5)	0.4005 (5)	0.3809 (5)	0.3809 (5)	0.3809 (5)	0.4005 (5)	0.3809 (5)	0.3809 (5)	0.3809 (5)	0.3809 (5)
	ds2c2sc13_E2	1 (5)	1 (5)	1 (5)	1 (5)	1 (5)	1 (5)	1 (5)	1 (5)	1 (5)	1 (5)	1 (5)
ds3c3sc6_E2	ds3c3sc6_E1	0.9069 (3)	0.9069 (3)	0.9069 (3)	0.9069 (3)	0.9069 (3)	0.9069 (3)	0.9069 (3)	0.9069 (3)	0.9069 (3)	0.9069 (3)	0.9069 (3)
	Frogs_MFCC_E1	0.5937 (6)	0.4789 (6)	0.4789 (6)	0.4789 (6)	0.4789 (6)	0.4789 (6)	0.5220 (5)	0.4789 (6)	0.4789 (6)	0.4789 (6)	0.4789 (6)
Frogs_MFCC_E3	Frogs_MFCC_E2	0.5937 (6)	0.5937 (6)	0.5937 (6)	0.5937 (6)	0.5937 (6)	0.5937 (6)	0.62135 (5)	0.5937 (6)	0.5937 (6)	0.5937 (6)	0.5937 (6)
	Golub_E1	0.5442 (3)	0.8897 (2)	0.2930 (4)	0.5442 (3)	0.5442 (3)	0.5465 (3)	0.5442 (3)	0.5465 (3)	0.5442 (3)	0.5084 (3)	0.5442 (3)
Leukemia_E2	Leukemia_E1	0.2859 (8)	0.2859 (8)	0.2859 (8)	0.2853 (8)	0.2853 (8)	0.2853 (8)	0.2853 (8)	0.2859 (8)	0.2853 (8)	0.2853 (8)	0.2853 (8)

As can be seen in Table 6.8 all MOEAs for the dataset `ds2c2sc13` when executed with the initial partitions of version 3, with 13 clusters, were also able to identify perfectly (ARI equals 1) version 2, with 5 clusters. The MOEAs applied in the dataset `ds3c3sc6` also had a good performance (ARI 0.9). `Frogs_MFCC` and `Golub` also surpasses an ARI of 0.5 in most cases. However, it was not identified an outstanding individual performance of a MOEA, all of them had similar performances in terms of recognition of different structures in a single set of solutions.

RQ2: Are the crossovers used in jMocle capable of contributing with the evolution the population according to the two objectives here considered?

The graphs in Figures 6.6 to 6.11 show the initial and final population for some representative datasets. In these graphs, it is possible to identify when the solutions of the final population preserved exclusively initial population individuals and when new solutions were created through the evolutionary process. It is also possible to observe which initial algorithms cover each area of the space corresponding to the two objective functions.

The partitions obtained with the initial algorithms are indicated by connected circles. The partitions contained in the final population of the jMocle executed with the MOEAs are indicated by markers with three axes. These results correspond to all partitions generated as output of 30 jMocle runs. When a three-axis marker is inside a circle then this element of the final population was selected from the initial population, when the three-axis marker is outside then this is a newly generated element that was not part of the initial population. The NSGA-II, SPEA2, and SPEA2+SDE MOEAs were selected to appear in the graphs based on the

performance they obtained in the hypervolume indicator which is directly reflected in how the fronts are arranged, facilitating visualization.

The points are distributed in the normalized space between 0 and 1 and the evolutionary process seeks to minimize the two objectives indicated in the graphics, connectivity and global deviation.

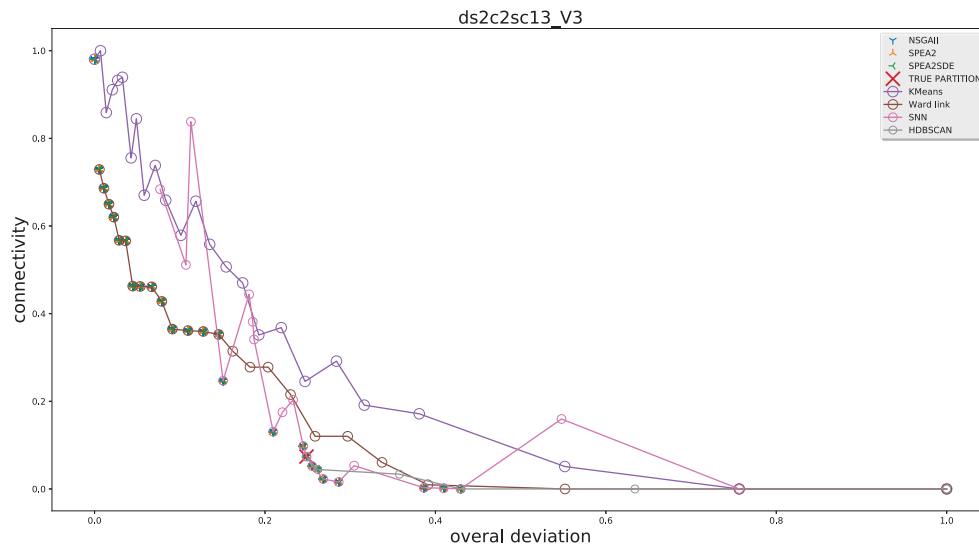


Figure 6.6: Initial population vs final population with HBGF crossover - ds2c2sc13_E3.

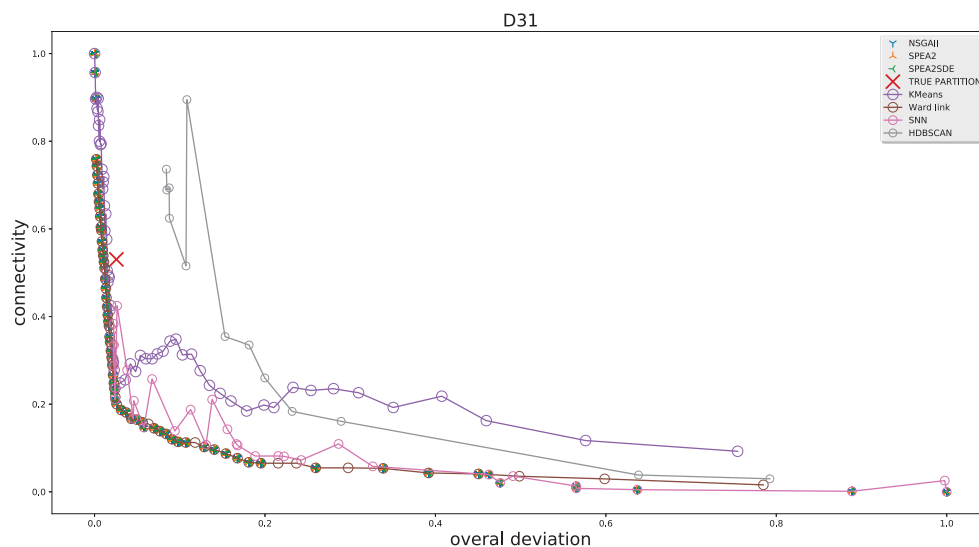


Figure 6.7: Initial population vs final population with HBGF crossover - D31.

In the artificial data sets ds2c2sc13 and D31 when using the HBGF crossover the results returned were limited to dominant individuals of the initial population. Indicated by $\times$ are the true partition of each dataset. In Figure 6.6, the initial population algorithms were able to generate partitions equivalent to true partition, with ARI=1, and the jMocle was able to return

these partitions among the set of solutions. The same does not occur in Figure 6.7 where the true partition is dominated by solutions from the initial population.

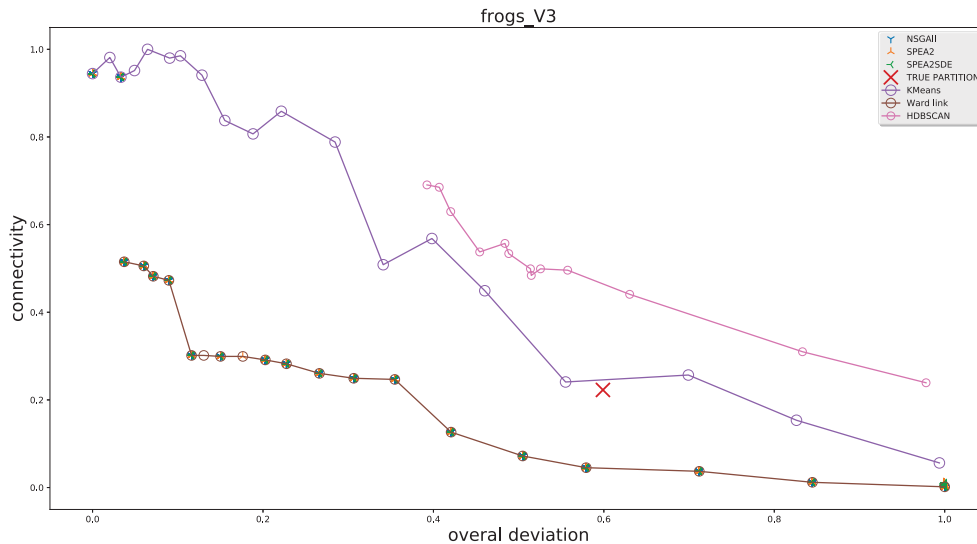


Figure 6.8: Initial population vs final population with HBGF crossover - Frogs_MFCC_E3.

Analyzing Figure 6.8 with the Frogs dataset it is observed that the Ward-linkage algorithm produced most of the solutions selected by the jMocle, k -means returned two solutions that presented better performance concerning the overall deviation, while the HDBSCAN has all its solutions dominated by other algorithms in the initial population. For this dataset, SNN was not able to generate any partition in the initial population.

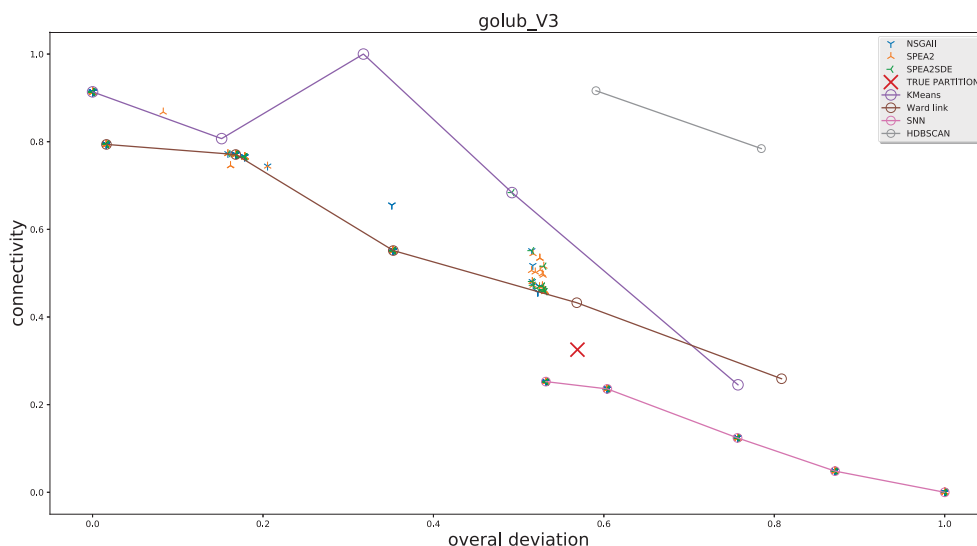


Figure 6.9: Initial population vs final population with HBGF crossover - Golub_E3.

In the real dataset Golub_E3, occurred the production of new solutions concentrated in a small region of the front. The other solutions were selected from the initial partitions obtained with the k -means, Ward-linkage and SNN algorithms.

Figures 6.10 and 6.11 represents the comparison between the initial and final populations of the MOEAs that obtained the best hypervolume with the crossover MCLA.

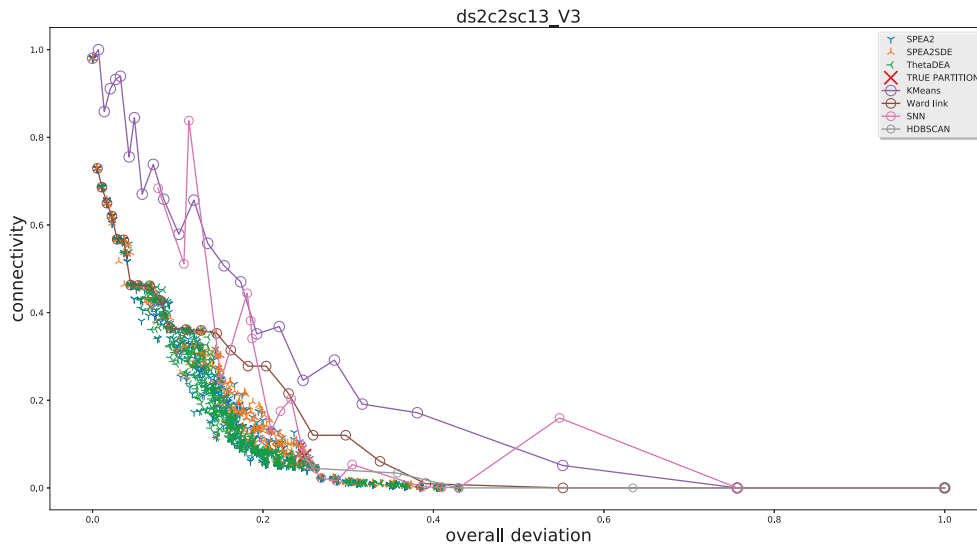


Figure 6.10: Initial population vs final population with MCLA crossover - ds2c2sc13_E3.

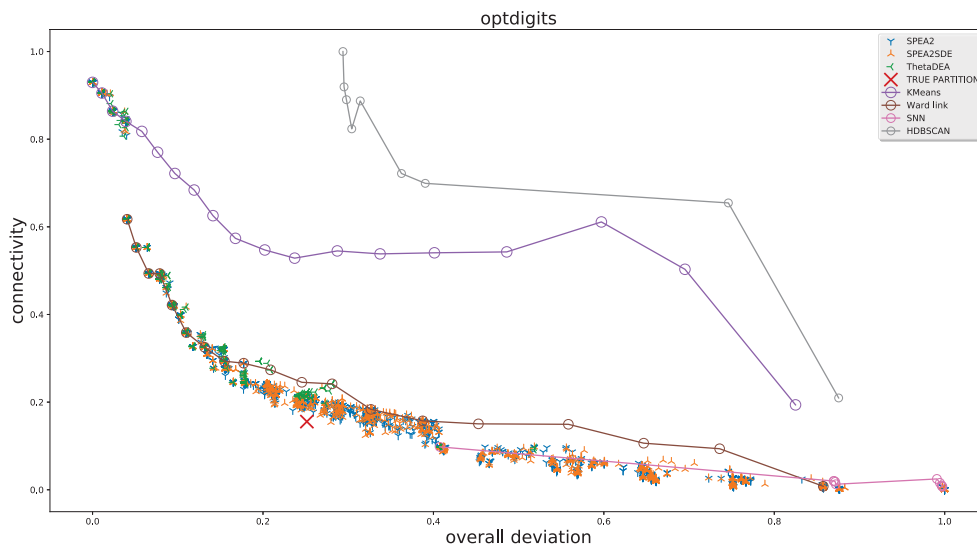


Figure 6.11: Initial population vs final population with MCLA crossover - OptDigits.

In the course of 30 jMocle executions, the MCLA crossover is capable of producing several new solutions (those not present in the initial population). These solutions are generally distributed along the front, composed of the dominant partitions of the initial population, meaning that the new solutions are as good as the best ones from the initial population when considering the two objectives.

Bearing this in mind, it can be observed that the set of solutions returned in the fronts of jMocle using the HBGF crossover in most cases were already part of the initial population, this

crossover had a low capacity to produce new solutions regardless of the MOEA used during the evolutionary phase. On the other hand, MCLA is capable of generating new solutions, however, these solutions are distributed along the front composed of the initial population dominant partitions.

This result characterizes that there was not enough evolution for the new population to completely dominate the initial population regardless of the MOEA used. A higher number of generations is considered further in **RQ3.1** with no impact in the evolution. A possible explanation to the poor evolution, regardless of the crossover selected, is that the initial algorithms already optimize clustering criteria and place many elements of the initial population in optimally local areas of the front.

RQ3: Is the initial population crucial to the performance of jMocle?

To answer this question, a random population of 100 individuals was generated for the dataset `ds3c3sc6_E2`. The number of clusters of each partition was selected randomly, varying between 2 and $2K^*$ with K^* being the true partition number of clusters, the distribution of the objects inside the clusters was also random. To ensure diversity the initial population size was defined to 100, doing so, for each k value there is a chance of existing approximately 9 different partitions inside the initial population, in a attempt to diversify the initial population.

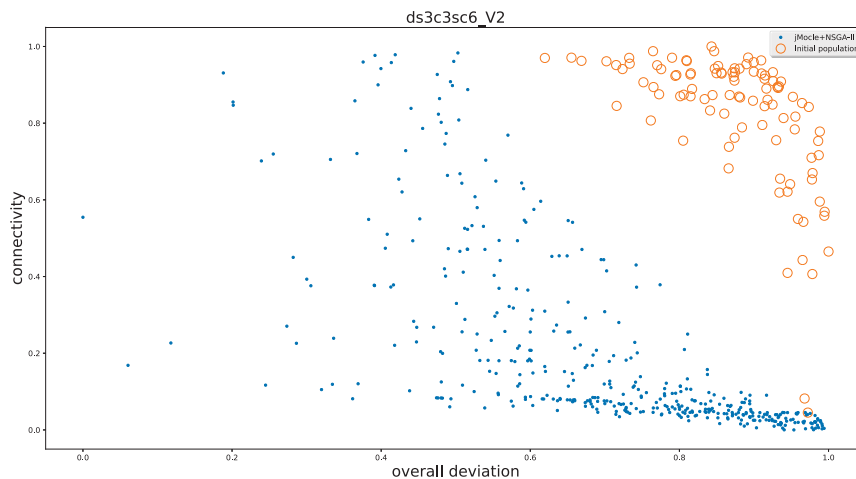


Figure 6.12: Random initial population - `ds3c3sc6_E2`.

We used the MCLA crossover that proved to be more efficient in generating new solutions during previous experiments. From Figure 6.12 it is possible to identify that the solutions returned in 30 executions dominate the initial population, indicating an evolution.

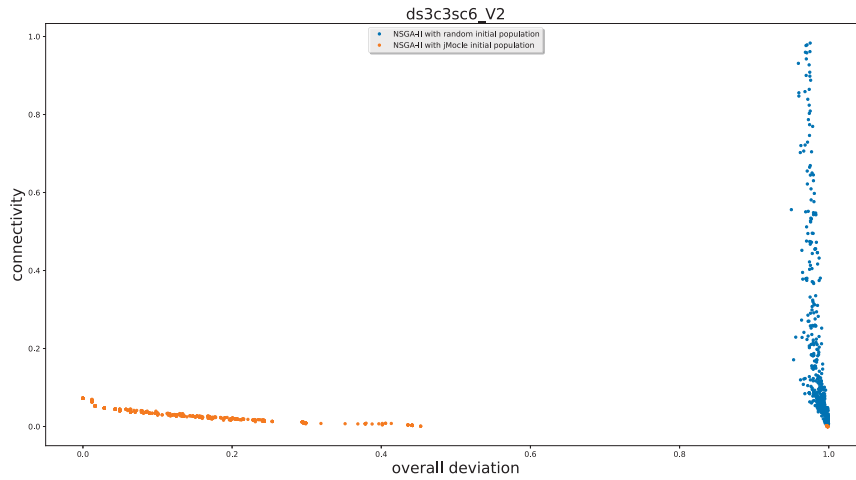


Figure 6.13: Random initial population vs jMocle initial population - ds3c3sc6_E2.

Figure 6.13 compares the results (final populations of 30 jMocle runs) evolved from the initial random population (blue dots) with results evolved from the population generated by the traditional clustering algorithms (orange dots). Although results with good connectivity were produced by both approaches, the initial population generated by clustering algorithms was fundamental to find partitions with good overall deviation.

RQ3.1: Are other component changes capable of improving jMocle's performance?

One of the tests performed to answer this question was to change an optimization parameter of jMocle, by doubling the number of iterations. We selected some MOEAs and some data sets as can be seen in Figure 6.9.

Table 6.9: Best ARI score in 30 executions for each problem

PROBLEM	MOEADD(50)	MOEADD(100)	NSGAII(50)	NSGAII(100)	SPEA2(50)	SPEA2(100)
D31	0.9534	0.9535	0.9534	0.9534	0.9534	0.9541
ds3c3sc6_E1	0.906	0.906	0.906	0.906	0.906	0.906
ds3c3sc6_E2	0.5852	0.5852	0.5852	0.5852	0.5852	0.5852
Golub_E1	0.9439	0.9439	0.5442	0.5442	0.5442	0.5442
Golub_E3	0.5444	0.5444	0.5444	0.5444	0.5444	0.5334
OptDigits	0.8267	0.8267	0.8136	0.8136	0.8267	0.8267
Seeds	0.7273	0.7610	0.7273	0.7273	0.7610	0.7484
tevc_20_60_1	0.6007	0.6007	0.6007	0.6007	0.6007	0.6007

In three cases there was a small ARI increase when using 100 generations, but in two occurrences the standard quantity of 50 generations was better, in the other cases there was a tie. These results indicate that increasing the number of generations does not provided a jMocle performance improvement.

Another recurring component in multi-objective clustering algorithms that are not used by MOCLE and jMocle is the mutation operator. Tests were performed using a random mutation operator with a $1/N$ probability of changing the cluster label to a different one within the range of possible clusters. It was also implemented in jMocle a centroid-based mutation using a $1/N$ probability that the object undergoing the mutation receives the label of the cluster corresponding to the nearest centroid. Both mutations were not able to produce any difference in the results analyzed.

An attempt was also made to use another crossover operator, for this purpose the CSPA ensemble technique described in Section 2.2.3 was used. The results obtained with this crossover method were similar to the HBGF crossover with the negative factor of CSPA being a slower technique due to performing the calculation of the co-association matrix.

Attempts to improve the results by increasing the number of generations, using simple mutation operators or using a different and simple crossover method have not produced a positive effect on the results.

RQ4: Are MOCK and Δ -MOCK better?

The evaluation made to answer this RQ considers the ARI obtained by the various algorithms to define which were able to return the partition with greater similarity to the true partition. For the multi-objective clustering algorithms, 30 executions are considered. Traditional clustering algorithms follow the protocol defined in Section 6.2.

In this case, HBGF and MCLA crossover are considered for jMocle. MOEA/DD and SPEA2 were the MOEAs chosen for reference, although they are equivalent to other MOEAs according to RQ2. Both versions of jMocle are compared with the clustering algorithms that originated the initial populations and also with the state-of-the-art multi-objective clustering techniques MOCK and Δ -MOCK.

Table 6.10: Best ARI score obtained from 30 jMocle runs in artificial datasets

problem	jMocle-H+MOEA/DD	jMocle-M+SPEA2	KMeans	Ward Link	SNN	HDBSCAN	Δ -MOCK	MOCK
D31	0.9535	0.9535	0.9535	0.9201	0.9464	0.806	0.7816	0.89
ds2c2sc13_E1	0.5292	0.3809	1.0	1.0	0.684	1.0	0.4271	0.3809
ds2c2sc13_E2	1.0	1.0	0.8752	0.8537	1.0	1.0	0.7285	1.0
ds2c2sc13_E3	1.0	1.0	0.6190	0.6264	1.0	1.0	0.8363	0.8724
ds3c3sc6_E1	0.9069	0.9348	0.8999	0.9069	0.7278	0.5145	0.6051	0.8816
ds3c3sc6_E2	0.5852	0.7445	0.5826	0.5852	0.6671	0.418	0.7101	0.7457
spiralsquare	0.9994	0.9994	0.4962	0.4986	0.9994	0.9902	0.9994	0.9994
tevc_20_60_1	0.6008	0.6017	0.5307	0.5855	0.9418	0.3035	0.9243	0.8278

Among the eight artificial data sets: in one case jMocle was able to return one partition better than the other algorithms; in 4 cases it returned the best partition already present in the database; in 1 case MOCK was able to find the best partition alone. Some insights for these results are better addressed in Chapter 7.

Table 6.11: Best ARI score obtained from 30 jMocle runs in real datasets

problem	jMocle-H+MOEA/DD	jMocle-M+SPEA2	KMeans	Ward Link	SNN	HDBSCAN	Δ -MOCK	MOCK
Frogs_MFCC_E1	0.4789	0.5044	0.522	0.4789	-	0.5022	0.2194	0.6168
Frogs_MFCC_E2	0.5937	0.6430	0.6213	0.5937	-	0.6218	0.6919	0.6856
Frogs_MFCC_E3	0.8156	0.8450	0.8403	0.8165	-	0.8494	0.9155	0.9093
Golub_E1	0.944	0.9088	0.944	0.6889	-0.0129	0.4441	0.9051	1.0
Golub_E3	0.5444	0.8921	0.687	0.4836	0.3098	0.4213	0.6645	0.9629
Leukemia_E1	0.2553	0.2553	0.2122	0.4469	0.2564	0.4469	0.2561	0.3125
Leukemia_E2	0.7871	0.7855	0.5219	0.5026	0.7871	0.5026	0.8305	0.8061
Libras	0.3448	0.3607	0.3217	0.3448	0.3376	0.1997	0.3933	0.4211
OptDigits	0.8267	0.8687	0.7455	0.8137	0.5999	0.6553	0.8472	0.9197
Seeds	0.7273	0.7388	0.7166	0.7273	0.6909	0.4256	0.4305	0.7388
UKCl	1.0	1.0	0.9914	0.9999	1.0	1.0	0.9974	0.9999

Analyzing the real data sets: in 6 cases the best partition among all executions was achieved by MOCK, in the case of *Seeds* dataset jMocle also found a similar partition to MOCK. Δ -MOCK was able to find in 3 cases the best partition among all executions and jMocle-M+SPEA2 found in 2 cases the best partitions.

With respect to the capacity of find the best partition in terms of ARI for the artificial datasets the algorithms were equivalent and for real datasets MOCK identified more frequently (6 occurrences) the best partition among all executions. However the overall values were close, and the considered measure for this RQ was just the best partition in terms of ARI. A more detailed

study about statistical differences and behaviour of each algorithm will be presented further in Chapter 7

RQ5: How jMocle performs in the optimization compared to the traditional MOCK and a version of the MOCK initialized with the same population as jMocle?

MOCK and MOCLE optimize the same objective functions, this allows the comparative analysis of the fronts obtained with both algorithms in the same graph. In this same comparison, we also used a version of the MOCK, here called MOCK-D, initialized with the same initial population as jMocle.

MOCK and MOCK-D have the parameters of interest L referring to the number of neighbors for the connectivity calculation, and the maximum number of clusters k_{max} . $L = 10$ and $k_{max} = \text{true partition's } k * 2$, following the same protocol of jMocle.

In the graphs are drawn solutions obtained after 30 jMocle runs using NSGA-II as MOEA compared to single MOCK and MOCK-D runs. To facilitate the visualization, jMocle executions with MCLA crossover is represented by unconnected points while the other algorithms are represented by connected fronts.

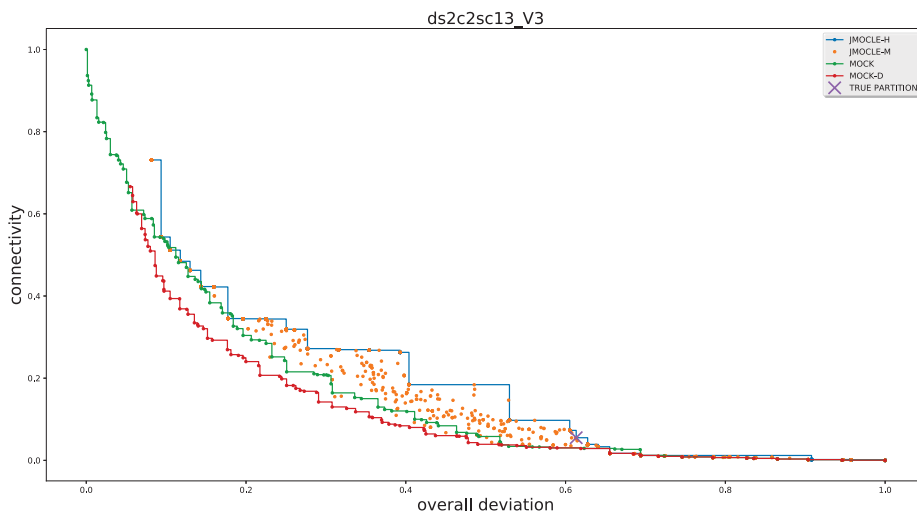


Figure 6.14: Comparison between MOCLE and MOCK/MOCK-D - ds2c2sc13_E3.

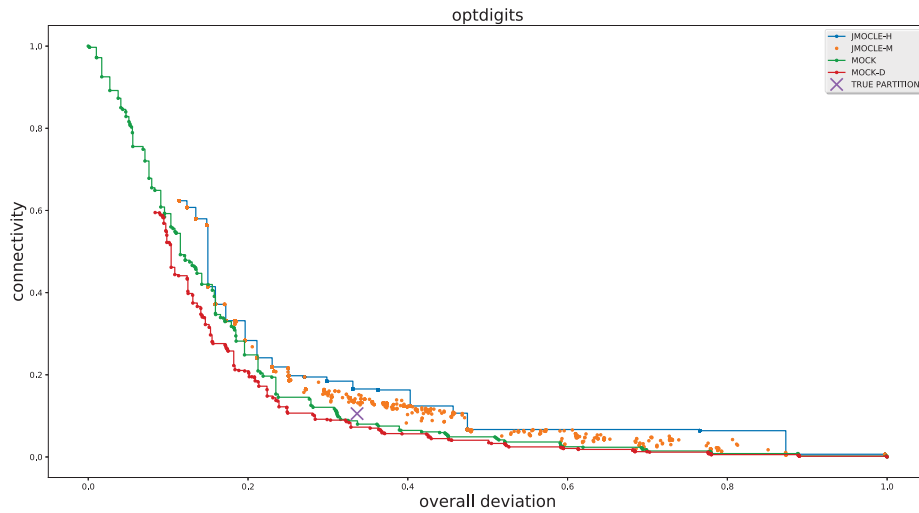


Figure 6.15: Comparison between MOCLE and MOCK/MOCK-D - OptDigits.

In Figures 6.14 and 6.15, it can be observed that the algorithm with the greatest capacity to evolve the population was the MOCK-D, followed by the MOCK, jMocle-M, and jMocle-H. In both cases, it can also be observed that the MOCK produced more solutions that optimize the overall deviation when compared to the other techniques.

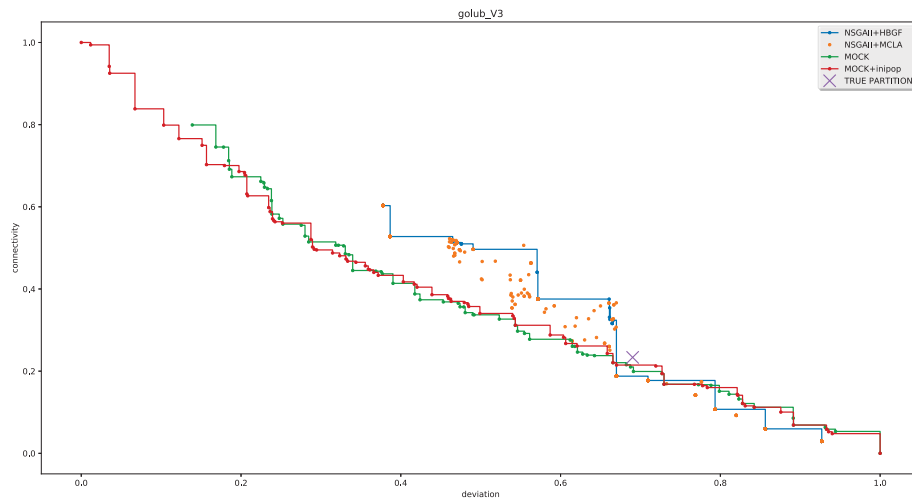


Figure 6.16: Comparison between MOCLE and MOCK/MOCK-D - Golub_E3.

The behavior of extending the front in the direction of minimizing the global deviation is also seen in Figure 6.16, in this case in both versions of the MOCK overcome the fronts of jMocle for this objective. In the cases of ds2c2sc13_E3 and Golub_E3 datasets, when taking into account the true partition, indicated by a $\times$ mark, we observe that minimizing overall deviation consequently generates solutions with worse connectivity and far from the true partition.

6.6 DISCUSSION

This chapter presented the first comparative analysis between the different MOEAs responsible for the optimization of clustering objective functions in multi-objective clustering. This analysis attempted to evaluate the performance of the algorithms among themselves, according to the ARI cluster validation criteria, the hypervolume indicator and other characteristics settled in the research questions.

The results obtained with the jMocle were compared with the algorithm MOCK and Δ -MOCK, state-of-the-art techniques of multi-objective clustering. The results did not point to evidence of statistical differentiation in the performance of different MOEAs in MOCLE, however, some substantial clues of the behavior of these algorithms can be traced, e.g., the further optimization that MOCK presented when using MOCLE's initial population. So in a search for more insights, in the next chapter, the comparison will continue but focused on the data-driven behavior of those algorithms.

7 DATA DRIVEN ANALYSIS

In this chapter, using a data-driven analysis, we will put MOCK and MOCLE into perspective with respect to Δ -MOCK. First will be treated about the datasets characteristics. Next presented the index used in the performance assessment. Then it will be described the experimental setup, ending with the results and discussion.

7.1 EXPERIMENTAL DESIGN

7.1.1 Datasets

Taken into account the discussion presented in the previous chapters, to have a better idea of performance of Δ -MOCK with respect to MOCK, as well as that of MOCLE, in this chapter we performed experiments with a subset of the datasets used in the previous chapter. It is a smaller but more diverse collection of datasets when compared to [36, 29, 30]. Table 7.1 summarizes the main characteristics of the 12 datasets.

Table 7.1: Dataset characteristics: number of objects n , number of attributes d , number of specified structures n^E , and number of clusters K for each structure E_i .

Type	Dataset	n	d	n^E	K
Artificial	D31	3100	2	1	31
	tevc_20_60_1	4395	20	1	60
	ds2c2sc13_E1	588	2	3	2
	ds2c2sc13_E2	588	2	3	5
	ds2c2sc13_E3	588	2	3	13
Real	Seeds	210	7	1	3
	Golub	72	3571	2	2, 3
	Leukemia	327	271	2	3, 7
	Libras	360	90	1	15
	OptDigits	5620	62	1	10
	Frogs_MFCC	7195	22	3	4, 8, 10
	UKC1	29463	2	1	11

D31, tevc_20_60_1 and ds2c2sc13 are synthetic datasets specially designed to contain several different properties. More specifically, D31 contains 31 equal-sized and spread clusters that are slightly overlapping and distribute randomly in a 2-dimensional space [71]. Due to the lack of spatial separation between clusters, this problem is hard to solve for algorithms based on optimizing connectedness or spatial separation. tevc_20_60_1 is one of the instances of the datasets used in [36] to assess the performance of Δ -MOCLE. It consists of 60 compact clusters, in 20-dimensional space, with arbitrary elongation and orientation which are hard to identify for algorithms based on cluster compactness. ds2c2sc13 was specially designed to contain three different structures: E1, E2 and E3 [29]. Figure 2.6 illustrates the existing underlying structures presented in this dataset. E1 represents two well-separated clusters, which can be found by techniques based on optimizing connectedness or compactness. In contrast, E2 and E3 combine different types of clusters that could be hard to find with techniques based only on connectedness or compactness. For example, in E2 there is a smile-shaped (non-convex)

cluster, an elongated cluster, and three globular clusters. In E3, the same smile-shaped cluster appears, but the elongated cluster of structure E2 can be viewed as three globular clusters. In addition, each of E2's globular clusters can be seen as three elongated clusters in E3.

Concerning the real datasets, we would like first to draw the attention to the fact that for most of them the underlying structure in the data (true partition) was found in the context of supervised learning. Thus, since cluster structures do necessarily is in agreement with the class labels, the performance of the clustering algorithms on these data can sometimes be low. Nevertheless, we included such datasets for they present different important characteristics. For example, *Golub* [37] and *Leukemia* [78] have a small number of objects (distributed in clusters of very different sizes), but a large number of attributes, typical of bioinformatics data. To test the behavior of the algorithms on data with a very large number of observations, we included *UKC1*. This is one of the real datasets used in [36] to evaluate the performance of Δ -MOCLE. *Seeds*, *Libras*, *OptDigits* and *Frogs_MFCC* (version with 10 classes) were obtained from the UCI Machine Learning Repository [22]. Whereas *Seeds*, *Libras*, *OptDigits* have clusters approximately of the same size, *UKC1_1* and *Frogs_MFCC* contains clusters ranging, respectively, from 1597 to 3000 objects and from 68 to 3478.

7.1.2 Performance assessment

The validation index used to measure the performance of the algorithms in this data-driven analysis was the ARI previously presented in Section 2.3.1, and is also one of the indices used in the evaluation of different MOEAs in MOCLE (see Section 6.5).

7.1.3 Experimental Setup

To execute *MOCK* and Δ -*MOCK*, we employed the same general settings as reported in [40, 36]: 1) Euclidean distance as distance function; 2) $L = 10$ nearest neighbors used during initialization, mutation, and in the definition of connectivity; 3) the maximum number k^{max} of clusters in the initial population was set to $2k^*$, where k^* is the number of clusters in the true partition representing the dataset. Concerning the representation of the solutions for Δ -*MOCK*, in this paper, we used the Δ -locus scheme for, as pointed out in [36], it tends to produce better results than the Δ -binary one. Finally, regarding the length of Δ -locus scheme, we set δ according to one of the heuristics employed in [36] in which the length of the encoding is defined as a function of $\sim 5\sqrt{n}$, where n is the number of objects in the dataset. This is one of the heuristics employed in [36] to chose the value of δ . Indeed, given that the initial population was generated by using Δ -*MOCK*'s initialization procedure, according to the authors, in most of the cases they analyzed, the setting of δ did not seem to affect the performance in terms of clustering recovery. To execute *MOCK*, we used the software available at <https://personalpages.manchester.ac.uk/staff/Julia.Handl/mock.html>. The implementation of Δ -*MOCK* is available at <https://github.com/garzafabre/Delta-MOCK>.

In terms of parameter configurations, for the case of MOCLE, we proceeded as follows: 1) NSGA-II as optimization method; 2) crossover operator: the cluster ensemble technique HBGF (Hybrid Bipartite Graph Formulation) [32]; 3) objective functions: overall deviation and connectivity as in *MOCK* [40]; 4) Euclidean distance as distance function; 5) 10 for the number L of nearest neighbors to calculate the connectivity. For the recombination probability, as well as the number of generations, we employed the same setting as in [29]. To generate the initial population, we used: k -means [63], the Ward linkage method [74], the shared nearest neighbor-based clustering (SNN) [24], and hierarchical density-based clustering (HDBSCAN) [12]. These algorithms have been chosen to provide a wide range of recovery effectiveness, as well as

to account for clustering criteria such as compactness (e.g., k -means and the Ward linkage method) and connectedness (e.g., SNN and HDBSCAN). More specifically, to be consistent with MOCK/ Δ -MOCK's initialization, we adjusted the parameters of such algorithms to produce partitions containing clusters in the range $\{2, 2k^*\}$. For example, k -means was executed 30 times for each k in $\{2, 2k^*\}$ and the partition with the smallest quadratic error was selected. In the case of the Ward linkage method, we generated the hierarchies and cut them to produce one partition for each value of k . As SNN and HDBSCAN find automatically the number k of clusters [24, 12], for each dataset, we run them with several different configurations of their hyper-parameters so as to try to yield partitions with k in $\{2, 2k^*\}$. Despite this, for SNN and HDBSCAN, in most of the datasets, we could not generate a set of partitions covering the complete range. For instance, for `Frogs_MFCC` SNN did not produce any partition in the required range. For all four algorithms, the Euclidean distance was selected as the proximity measure. To run SNN, we used the implementation of its authors and for k -means and the Ward linkage method we employed Scikit-learn [66]. To execute HDBSCAN, we used the Python library available at <https://github.com/scikit-learn-contrib/hdbscan#egg=hdbscan>.

Finally, as MOCK, Δ -MOCK, and MOCLE are non-deterministic, we execute the experiments 30 times. All the results (set of solutions) generated with the application of MOCK, Δ -MOCK, and MOCLE to all datasets, as well as the initial populations used by MOCLE, are available at <https://github.com/kultzak/multiobjective-clustering-datadriven-analysis>.

7.2 RESULTS AND DISCUSSION

This Section investigates the ability of MOCLE, MOCK and Δ -MOCK to produce high-quality clustering solutions as measured by the ARI, as well as issues concerning the numbers of clusters of the best partitions and the size of the sets of solutions. To do so, we proceeded as follows. For each of the 30 runs of each algorithm, we calculated the ARI between each partition π_i in the set Π_F of solutions produced by the algorithm and the known structure π^* of the corresponding dataset. Then, we selected the partition in Π_F with the best ARI. Next, we calculated, over the 30 runs, the mean and standard deviation for the ARI of these best partitions, as well as for their number of clusters. These results are summarized in Table 7.2. In this table, for each dataset, the average of the best ARI obtained is highlighted in boldface and the average of the numbers of clusters is indicated in parenthesis.

7.2.1 Clustering performance

Looking at Table 7.2, one can observe that, overall, across the different datasets, MOCLE, MOCK and Δ -MOCK presented similar results concerning the ARI. Indeed, we performed the Friedman test [33] (significance level set to 0.05) in which we considered the algorithms as treatments being compared and the datasets as blocks. In this way, we tested the null hypothesis H_0 : there is no difference among the algorithms (ARI), against H_1 : there is a significant statistical difference between at least two algorithms. The result of the test showed that there is not enough evidence to reject H_0 .

Although there is no statistical evidence for a difference in performance (ARI) of the algorithms investigated, a more detailed analysis of the results presented for certain datasets can give some useful insights. We will first turn our attention to the synthetic datasets. For example, as discussed in Section 7.1.1, `D31` contains several compact globular (Gaussian) clusters that are slightly overlapping (see Figure 6.7). Thus, one would expect that any algorithm that optimizes

Table 7.2: ARI (number of clusters) of the best partition found by MOCLE, MOCK and Δ -MOCK. Average of 30 executions.

Dataset	MOCLE	MOCK	Δ -MOCK
D31	0.951 $\pm$ 0.003 (31.3 $\pm$ 0.5)	0.851 $\pm$ 0.026 (43.5 $\pm$ 5.9)	0.761 $\pm$ 0.024 (28.6 $\pm$ 2.3)
tevc_20_60_1	0.601 $\pm$ 0.000 (66.0 $\pm$ 0.0)	0.798 $\pm$ 0.027 (84.9 $\pm$ 8.1)	0.862 $\pm$ 0.035 (62.5 $\pm$ 3.0)
ds2c2sc13_E1	0.529 $\pm$ 0.000 (4.0 $\pm$ 0.0)	0.381 $\pm$ 0.000 (5.0 $\pm$ 0.0)	0.352 $\pm$ 0.000 (6.0 $\pm$ 0.0)
ds2c2sc13_E2	1.000 $\pm$ 0.000 (5.0 $\pm$ 0.0)	1.000 $\pm$ 0.000 (5.0 $\pm$ 0.0)	0.952 $\pm$ 0.000 (6.0 $\pm$ 0.0)
ds2c2sc13_E3	1.000 $\pm$ 0.000 (13.0 $\pm$ 0.0)	0.871 $\pm$ 0.013 (11.1 $\pm$ 0.2)	0.872 $\pm$ 0.000 (11.0 $\pm$ 0.0)
Seeds	0.727 $\pm$ 0.000 (4.0 $\pm$ 0.0)	0.711 $\pm$ 0.018 (3.2 $\pm$ 0.5)	0.672 $\pm$ 0.039 (3.6 $\pm$ 0.6)
Golub	0.512 $\pm$ 0.034 (3.0 $\pm$ 0.0)	0.854 $\pm$ 0.095 (2.4 $\pm$ 0.8)	0.644 $\pm$ 0.02 (6.0 $\pm$ 0.6)
Leukemia	0.785 $\pm$ 0.000 (8.0 $\pm$ 0.0)	0.778 $\pm$ 0.010 (10.3 $\pm$ 1.7)	0.771 $\pm$ 0.012 (11.0 $\pm$ 2.0)
Libras	0.345 $\pm$ 0.000 (24.0 $\pm$ 0.0)	0.395 $\pm$ 0.015 (15.6 $\pm$ 3.9)	0.386 $\pm$ 0.005 (15.1 $\pm$ 1.5)
OptDigits	0.814 $\pm$ 0.000 (13.0 $\pm$ 0.0)	0.884 $\pm$ 0.018 (22.9 $\pm$ 4.3)	0.834 $\pm$ 0.013 (14.9 $\pm$ 1.5)
Frogs_MFCC	0.816 $\pm$ 0.000 (6.0 $\pm$ 0.0)	0.876 $\pm$ 0.023 (15.9 $\pm$ 2.7)	0.902 $\pm$ 0.006 (23.2 $\pm$ 2.3)
UKC1	1.000 $\pm$ 0.000 (11.0 $\pm$ 0.0)	0.999 $\pm$ 0.002 (12.0 $\pm$ 1.0)	0.996 $\pm$ 0.001 (13.7 $\pm$ 1.0)

compactness such as k -means and Ward linkage method should be able to yield a partition with large ARI as can be seen in Table 7.3. Indeed, MOCLE found, on average, high-quality partitions with ARI compatible to that returned by k -means (0.951 against 0.953) — Figure 7.1 illustrate the median best partition found by MOCLE. On the other hand, on average, MOCK and Δ -MOCK yielded best solutions with much smaller ARI (respectively, 0.851 and 0.761). The main reason for the poor performance of Δ -MOCK on D31 could be in the procedure that it employs to create the initial population. As seen in Section 4.2, for Δ -MOCK, the initial population is created by removing *interesting* links from an MST built on the data. A link between two objects is considered *interesting* if neither of them is part of the other object’s set of L nearest neighbors. In the case of D31, as two clusters overlap the density at the boundary of the clusters increases. As consequence objects in this region might have several nearest neighbors in common. This way, according to the strategy used, the links between objects in such regions would not be classified as *interesting* and, consequently, they would not be potential candidates to be removed and the clusters would remain merged. Indeed, a visual inspection of the median best solutions obtained with Δ -MOCK for D31 shows that several clusters of the true partition were merged forming chain-like clusters (Figure 7.2). MOCK uses also a strategy similar to that of Δ -MOCK (see Section 4.1), but it has another phase in which solutions generated from k -means are added to the initial population. In this case of globular clusters, even in the presence of overlap, the use of k -means within the strategy of generating the initial population of MOCK could have provided a competitive advantage over the Δ -MOCK. For example, a visual inspection of one of the median best solutions obtained with MOCK shows that, in most cases, the clusters were not merged (Figure 7.3). On the other hand, as can be seen in the figure, it found clusters with down to two objects (e.g., cluster 40 on the bottom of the left corner) — this behavior will be discussed in Section 7.2.2.

In contrast to the result for D31, for tevc_20_60_1, MOCK and Δ -MOCK showed much larger average ARI (respectively, 0.798 and 0.862) than that of MOCLE (0.601). Indeed, in its initial population (see Table 7.3) MOCLE had a high-quality partition (ARI equals to 0.942) generated by the SNN, however such a partition was not kept in the set of final solution as it was dominated by one of the partitions generated by the Ward linkage method. Furthermore, in this specific case, the use of a cluster ensemble technique as crossover did help to keep a significative part of the SNN’s solution in the population. With respect to the difference between the performance of MOCK and Δ -MOCK, this was an expected result in that, as explained in

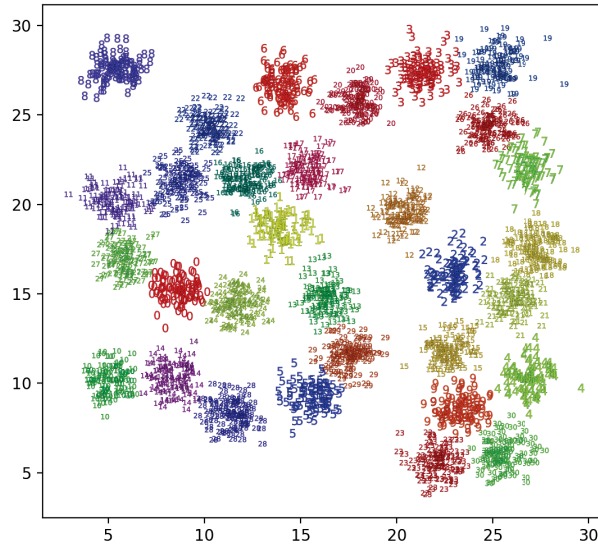


Figure 7.1: D31: median best partition found by MOCLE ($ARI = 0.953$ and $k = 31$). Points with the same marker (1,...,31) belong to the same cluster.

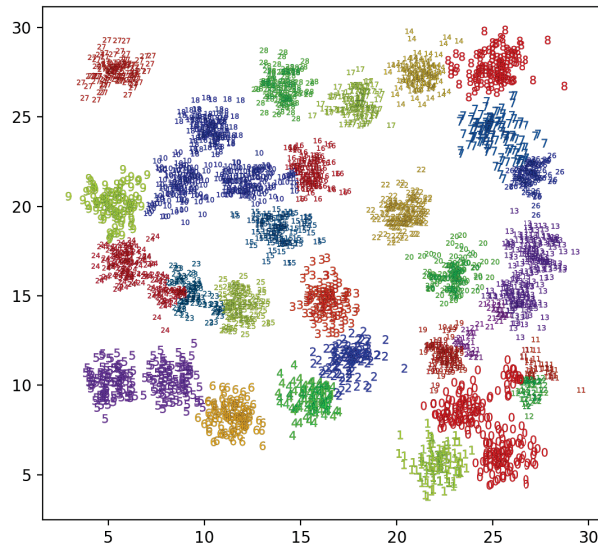


Figure 7.2: D31: median best partition found by Δ -MOCK ($ARI = 0.757$ and $k = 29$). Points with the same marker (1,...,29) belong to the same cluster.

Section 4, `tev_c20_60_1` is an instance of a collection of 350 datasets especially designed to test the gain of performance of the new characteristics of the Δ -MOCK in relation to the MOCK.

Concerning `ds2c2sc13` (Figure 2.6), our experimental results show that their structures present some challenges in what concerns the optimization of the objective functions, mainly the connectivity. Depending on the value of the number L of nearest neighbors used in the computation of the connectivity, one can detect the two well-separated clusters in $E1$ but fail to identify the small clusters present in $E3$ and vice-versa. Here, as in [40, 36], we set L to 10. More specifically, for $E1$, MOCLE, MOCK and Δ -MOCK could not find the two well-separated clusters presented in this structure. This happened for the true partition, π^* , was, in terms of the values of the objective functions, dominated by other partitions whose values for the connectivity was optimal as that of π^* ($con = 0$), but with smaller overall deviation/intra-cluster variance (these partitions had a larger number of clusters than π^*). This result contrasts to that in [29] in which with $L = 30$ (5% of the number of objects in the dataset) both MOCLE and MOCK

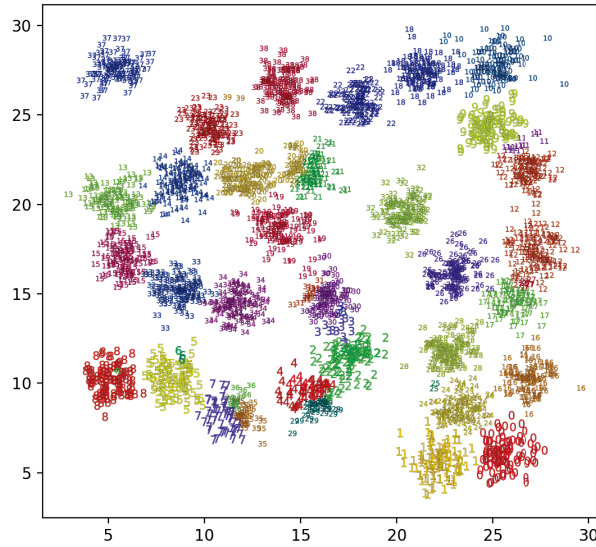


Figure 7.3: D31: median best partition found by MOCK ($ARI = 0.860$ and $k = 41$). Points with the same marker (1,...,41) belong to the same cluster.

Table 7.3: ARI (number of clusters) of the best partition found by k -means, the Ward linkage method, SNN and HDBSCAN.

Dataset	KMeans	Ward-linkage	SNN	HDBSCAN
D31	0.953 (31)	0.920 (31)	0.946 (32)	0.806 (31)
tevc_20_60_1	0.531 (115)	0.586 (109)	0.942 (113)	0.303 (76)
ds2c2sc13_E1	1.000 (2)	1.000 (2)	0.684 (3)	1.000 (2)
ds2c2sc13_E2	0.875 (5)	0.854 (8)	1.000 (5)	1.000 (5)
ds2c2sc13_E3	0.619 (18)	0.626 (20)	1.000 (13)	1.000 (13)
Seeds	0.717 (3)	0.727 (4)	0.691 (3)	0.426 (3)
Golub	0.944 (2)	0.689 (2)	-0.013 (2)	0.444 (3)
Leukemia	0.785 (8)	0.787 (8)	0.522 (6)	0.503 (7)
Libras	0.322 (19)	0.345 (24)	0.338 (27)	0.200 (13)
OptDigits	0.745 (11)	0.814 (13)	0.600 (16)	0.655 (10)
Frogs_MFCC	0.840 (5)	0.816 (6)	-*	0.849 (21)
UKC1	0.991 (11)	1.000 (11)	0.999 (13)	1.000 (11)

found E1. For structure E2, like in [29], both MOCLE and MOCK found, in each of the 30 runs, a partition that corresponded to the ground truth π^* , that is, an ARI equals to 1. On the other hand, the best partition found by Δ -MOCK in each of the runs had an ARI equals to 0.952 and contained six clusters: one of the large globular clusters was split into two. In [29], with L set to 30, neither MOCK nor MOCLE found E3. In contrast, here, in all its runs, MOCLE did find a partition corresponding to the true structure in E3 (ARI equals 1). For the case of MOCK and Δ -MOCK the most frequent best partition in the 30 runs merged two pairs of close clusters (refinement of the globular clusters in E2). In summary, as discussed in Section 4.1, depending on the value of L and the underlying structure in the data (e.g., a cluster with fragmented dense regions), in terms of connectivity, this could mean that several different solutions will have an optimal value ($con = 0$) and, as a consequence, decision will be taken essentially based on the overall deviation.

Now, turning our attention to the real datasets, in terms of poor performance according to the ARI, there is a case that stands out: the results of MOCLE and Δ -MOCK for Golub (respectively, 0.512 and 0.644) when compared to that of MOCK (0.854). In the case of

MOCLE, as it happened for `tevc_20_60_1`, in its initial population (see Table 7.3) there was a high-quality partition (ARI equals to 0.944) generated by k -means, yet such a partition was not kept in the set of final solution for it was dominated by one of the partitions generated by the SNN. In fact, as previously discussed, the good performance obtained with k -means for `Golub` could explain why MOCK performed much better than Δ -MOCK, as MOCK initial population is built with not only MST-derived partitions like the Δ -MOCK, but also with partitions generated with k -means.

7.2.2 Number/size of clusters of the best partitions

According to the results presented in Table 7.2, for most of the datasets, MOCLE, MOCK and Δ -MOCK returned best partitions containing a number of clusters equals or close to that (k^*) of the corresponding true partition (π^*). The exceptions were the following cases: 1) MOCK for `OptDigits` whose partitions on average have more than 22 clusters ($k^* = 10$) and Δ -MOCK for `Frogs_MFCC` with partitions containing more than 23 clusters ($k^* = 10$).

Although the average number of clusters of the best partitions, across the different datasets, as well as the value of ARI found by the algorithms investigated are similar, a more detailed analysis of the sizes of the clusters in the partitions can give some useful insights. To do so, for each dataset and algorithm, we chose to inspect the best, in terms of ARI, median partition, that is, the best partition whose ARI is the middle value among those of 30 best partitions. As a result of this analysis we identified that for most datasets, MOCK and Δ -MOCK generated partitions containing clusters having as few as five objects. Indeed, for `Golub` and `Leukemia` the partitions produced by Δ -MOCK presented more than one cluster containing a single object. MOCK also for the case of `Leukemia` generated a partition containing some clusters with only two objects. Likewise, for `OptDigits`, both algorithms yielded partitions with some clusters having two to five observations. For `Frogs_MFCC` and `UKC1`, Δ -MOCK produced partitions with some clusters presenting, respectively, seven and 17 objects (for the case of `UKC1`, which consists of 11 well-separated clusters, the smallest cluster in true partition has 1597 objects).

The main reason for the problem described in the previous paragraph is the value of the parameter $L = 10$ (number of nearest neighbors) used by MOCK and Δ -MOCK for both the computation of the measure DI (generation of initial population) and the connectivity (objective function). For example, an inspection of the size of the clusters in the partitions forming the initial populations showed that several of them contained clusters having down to one object. This means that DI, computed based on $L = 10$ nearest neighbors, for most of the datasets analyzed, was not effective in avoiding the isolation of outliers. Furthermore, as the connectivity is also computed based on the L nearest neighbors, to a certain extent, this creates, during evolution, a bias towards solutions already present in the initial population. In fact, as pointed out in [40], in the case of MOCK (for Δ -MOCK also), the definition of the parameter L determines the algorithm's sensitivity to small clusters — not only during evolution by means of the computation of the connectivity, but most importantly in the generation of the initial population. There is a trade-off in that the choice of a “large” value is needed to prevent outlier clusters, but if L is too “large” some existing “small” clusters might not be detected. As mentioned in Chapter 4, in the experiments conducted in [40, 36] the authors set L to 10 and obtained adequate performance, in terms of ARI, across the range of datasets studied. However, differently from the study presented here, they did not performed an analysis of the sizes of the clusters in the partitions produced.

7.3 DISCUSSION

In this chapter we performed a data-driven analysis of three closely related multi-objective clustering algorithms: MOCLE, MOCK and Δ -MOCK. More specifically, we analyzed, based on the results of experiments conducted on five synthetic and seven real datasets, the performance of these algorithms in terms of 1) their ability to recover the true partition existing in the data, 2) the numbers of clusters in the partitions of the solution set, and 3) the size of the clusters.

As discussed in Section 7.2.1, the results of our analysis showed that, for the collection of datasets analyzed, according to the Friedman test (significance level of 0.05), there is no statistical evidence that, in terms of the adjusted Rand index (ARI), one algorithm could be considered better than the other. This is, by itself, an interesting result in that, so far, in the literature, the experimental studies comparing MOCK and MOCLE did not support their findings on a hypothesis test [29, 30, 31]. In the case of MOCK and Δ -MOCK, in the comparison presented in [36] the authors did perform a hypothesis test and found that the performance of Δ -MOCK was superior of that of MOCK. However, as discussed in Section 4.2, differently from the analysis conducted in this paper, the collection of datasets used in [36], though containing more than 350 datasets, does not present much diversity limiting on the extent to which their results can be generalized.

Also, as discussed in Section 7.2.2, despite the quality in terms of ARI, for most datasets, MOCK and Δ -MOCK generated partitions containing clusters having as few as five objects. For instance, for `Golub` the median best partition produced by Δ -MOCK presented more than one cluster containing a single object. Likewise, for `OptDigits`, both algorithms yielded median best partitions with some clusters having two objects. Indeed, the problem of partitions with outlier clusters was present already in the initial population. In other words, DI computed based on $L = 10$ nearest neighbors, for most of the datasets analyzed, was not effective in avoiding the isolation of outliers. Furthermore, as the connectivity was also computed based on $L = 10$, these partitions with outlier clusters made to the set of final solutions.

8 CONCLUSION AND FUTURE WORK

In this study we explored the behavior of different multiobjective algorithms in clustering, primarily investigating MOCLE, MOCK and Δ -MOCK. We choose MOCLE as basis to analyze the impact of using several MOEAs available in the jMetal framework, the so-called jMocle approach did not present evidence to support statistical differences between the MOEAs performance. Some characteristics of the clustering techniques can influence this result, as the high-quality initial population produced by these algorithms with initial solutions placed in possibly locally optimum areas.

Analyzing MOCK when performing a slight modification to use MOCLE's initial population (produced with traditional clustering algorithms) can give us a clue about the importance of the initialization procedure. In these cases, MOCK performed equal or better than using its original initialization routine. While using a random initial population, in that case in MOCLE, the results obtained were poor, reinforcing the importance of initialization.

In most cases the fronts obtained with MOCK dominate the results of jMocle, indicating that the MOCK further optimizes the objectives. However, considering the optimization aspect it is important to note that often the structures previously known in the data sets are dominated by solutions already present in the jMocle's initial population. In such cases, optimizing connectivity and overall deviation can have the deleterious effect of pushing the set of solutions away from what would be the ideal solution.

Although there is no statistical evidence for difference in performance (ARI) of the multiobjective clustering algorithms investigated, a more detailed analysis of the results observed for certain datasets gave us some useful insights on their strengths and weaknesses, mainly concerning MOCK and Δ -MOCK. Let's first turn our attention to issues regarding the strategy to generate the initial population. As pointed out in [36], for MOCK and Δ -MOCK, given the low mutation rates normally employed, the generation of new candidate solutions during optimization relies basically on recombination. Thus, in this context, the partitions introduced during initialization plays a key role in restricting the region of the solution space where the search will be performed. This holds also true for the case of MOCLE, by not applying mutation, it relies upon the initialization and ensemble schemes.

More specifically, concerning MOCK, according to [36], in terms of the strategy to generate the initial population, the largest contributions to performance are attained by the first phase (approach based on the measure *degree of interestingness* (DI) and the MST), whereas the second phase (solutions generated from k -means) is responsible for a significant computational overhead. Motivated by this, in Δ -MOCK they employed a strategy based only on the first phase of MOCK. However, as discussed in Section 7.2.1, such a DI-based only strategy utilized by Δ -MOCK cannot deal efficiently with data whose clusters present some degree of overlap. In contrast, at least for the case of "Gaussian" clusters, because of the partitions generated with k -means, such a problem is less pronounced for the case of MOCK.

8.1 FUTURE WORK

The fact that some true partitions have been dominated by individuals in the initial populations gives us scope to explore the use of other objective functions. The trade-off between overall deviation and connectivity in those cases was not enough to match the exact conformation in data, so the initial partitions ended optimizing these objectives to a point detrimental to the results in

terms of ARI. The development, use or combination of more objective functions, therefore can be useful in such cases.

For the case of MOCK and Δ -MOCK, the interaction between the use of L for both generating the initial population and calculating the connectivity should be investigated further so as to provide more sound guidelines to help to choose the value of such a parameter. Another issue worth investigating further is the limitations of using connectivity (con) as objective functions. As discussed in Section 4.1 and in Section 7.2.1, depending on the value chosen for L , several different partitions could present the same optimal value for the connectivity ($con = 0$). In a scenario like this, the connectivity would not be much discriminative, and the decision would be essentially taken based on the overall deviation.

One more interesting research direction is to investigate, for example, to what extent MOCLE can benefit from the use of base partitions generated by the initialization procedure employed by Δ -MOCK. As discussed in Section 4.3, currently MOCLE's initial population (set of base partitions) is built by running several conceptually different mono-objective clustering algorithms. Likewise, given the limitations of the initialization strategies used by MOCK and Δ -MOCK such as, for instance, the problem of overlapping clusters, one direction possible would be, like in MOCLE, to add partitions generated by existing mono-objective clustering that can deal with overlap and clusters with different densities. Here, as pointed out in [36], we would have to take into account the trade-off between the computational overhead of running the base algorithm and the gain in terms of quality of the set of solutions generated. Indeed, besides the issue regarding the generation of the initial population, the three algorithms analyzed here have interesting approaches to solve the clustering problem that could be combined to provide more robust solutions, but how to combine these ideas is an open question that needs to be investigated.

REFERENCES

- [1] Charu C. Aggarwal and Chandan K. Reddy. *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition, 2013.
- [2] Vanessa Antunes, Tiemi C. Sakata, Katti Faceli, and Marcilio C.P. de Souto. Hybrid strategy for selecting compact set of clustering partitions. *Applied Soft Computing*, 87:105971, 2020.
- [3] Giuliano Armano and Mohammad Reza Farmani. Multiobjective clustering analysis using particle swarm optimization. *Expert Systems with Applications*, 55:184–193, 2016.
- [4] Johannes Bader, Kalyanmoy Deb, and Eckart Zitzler. Faster hypervolume-based search using monte carlo sampling. In *Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems*, pages 313–326. Springer, 2010.
- [5] Johannes Bader and Eckart Zitzler. Hype: An algorithm for fast hypervolume-based many-objective optimization. *Evolutionary computation*, 19(1):45–76, 2011.
- [6] D Barbara. An introduction to cluster analysis for data mining. *Retrieved March*, 13:2006, 2000.
- [7] Nicola Beume, Boris Naujoks, and Michael Emmerich. Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 2007.
- [8] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. PISA — a platform and programming language independent interface for search algorithms. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, pages 494 – 508, Berlin, 2003. Springer.
- [9] Tossapon Boongoen and Natthakan Iam-on. Cluster ensembles: A survey of approaches with recent extensions and applications. *Computer Science Review*, 28:1–25, 2018.
- [10] Dimo Brockhoff and Eckart Zitzler. Are all objectives necessary? on dimensionality reduction in evolutionary multiobjective optimization. In *Parallel Problem Solving from Nature-PPSN IX*, pages 533–542. Springer, 2006.
- [11] Edmund K Burke, Graham Kendall, et al. *Search methodologies*. Springer, 2005.
- [12] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer, 2013.
- [13] Sanjay Chakraborty, NK Nagwani, and Lopamudra Dey. Performance comparison of incremental k-means and incremental dbSCAN algorithms. *arXiv preprint arXiv:1406.4751*, 2014.

- [14] CA Coello Coello, G Toscano Pulido, and E Mezura Montes. Current and future research trends in evolutionary multiobjective optimization. In *Information processing with evolutionary algorithms*, pages 213–231. Springer, 2005.
- [15] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.
- [16] David W Corne, Nick R Jerram, Joshua D Knowles, and Martin J Oates. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 283–290, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- [17] Indraneel Das and John E Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.
- [18] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601, 2014.
- [19] Kalyanmoy Deb, Kaisa Miettinen, and Shamik Chaudhuri. Toward an estimation of nadir objective vector using a hybrid of evolutionary and local search approaches. *IEEE Transactions on Evolutionary Computation*, 14(6):821–841, 2010.
- [20] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.
- [21] Kalyanmoy Deb and Dhish Kumar Saxena. On finding pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems. *Kangal report*, 2005011, 2005.
- [22] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.
- [23] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.
- [24] Levent Ertoz, Michael Steinbach, and Vipin Kumar. A new shared nearest neighbor clustering algorithm and its applications. In *Workshop on clustering high dimensional data and its applications at 2nd SIAM international conference on data mining*, pages 105–115, 2002.
- [25] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.
- [26] Katti Faceli. *Um framework para análise de agrupamento baseado na combinação multi-objetivo de algoritmos de agrupamento*. PhD thesis, Universidade de São Paulo, 2006.
- [27] Katti Faceli, André C. P. L. F. de Carvalho, and Marcílio C. P. de Souto. Multi-objective clustering ensemble with prior knowledge. In Marie-France Sagot and Maria Emilia M. T. Walter, editors, *Advances in Bioinformatics and Computational Biology*, pages 34–45, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

- [28] Katti Faceli, André Carlos Ponce de Leon Ferreira de Carvalho, and Marcílio Carlos Pereira de Souto. Multi-objective clustering ensemble. In *6th International Conference on Hybrid Intelligent Systems (HIS 2006), 13-15 December 2006, Auckland, New Zealand*, page 51, 2006.
- [29] Katti Faceli, André Carlos Ponce de Leon Ferreira de Carvalho, and Marcílio Carlos Pereira de Souto. Multi-objective clustering ensemble. *International Journal of Hybrid Intelligent Systems*, 4(3):145–156, 2007.
- [30] Katti Faceli, Marcilio CP de Souto, Daniel SA de Araújo, and André CPLF de Carvalho. Multi-objective clustering ensemble for gene expression data analysis. *Neurocomputing*, 72(13-15):2763–2774, 2009.
- [31] Katti Faceli, Tiemi C. Sakata, Marcílio Carlos Pereira de Souto, and André Carlos Ponce de Leon Ferreira de Carvalho. Partitions selection strategy for set of clustering solutions. *Neurocomputing*, 73(16-18):2809–2819, 2010.
- [32] Xiaoli Zhang Fern and Carla E Brodley. Solving cluster ensemble problems by bipartite graph partitioning. In *Proceedings of the twenty-first international conference on Machine learning*, page 36. ACM, 2004.
- [33] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.
- [34] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
- [35] Guojun Gan, Chaoqun Ma, and Jianhong Wu. *Data clustering: theory, algorithms, and applications*, volume 20. Siam, 2007.
- [36] Mario Garza-Fabre, Julia Handl, and Joshua Knowles. An improved and more scalable evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, 22(4):515–535, 2017.
- [37] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.
- [38] Raquel Hernández Gómez and Carlos A Coello Coello. Mombi: A new metaheuristic for many-objective optimization based on the r2 indicator. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2488–2495. IEEE, 2013.
- [39] Julia Handl and Joshua Knowles. Multiobjective clustering with automatic determination of the number of clusters. *Technical Report*, 2004.
- [40] Julia Handl and Joshua Knowles. An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, 11(1):56–76, 2007.
- [41] Julia Handl and Joshua D. Knowles. Exploiting the trade-off - the benefits of multiple objectives in data clustering. In *Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005, Guanajuato, Mexico, March 9-11, 2005, Proceedings*, pages 547–560, 2005.

- [42] Julia Handl and Joshua D. Knowles. Multi-objective clustering and cluster validation. In *Multi-Objective Machine Learning*, pages 21–47. 2006.
- [43] Zhenan He, Gary G Yen, and Jun Zhang. Fuzzy-based pareto optimality for many-objective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 18(2):269–285, 2014.
- [44] Raquel Hernández Gómez and Carlos A Coello Coello. Improved metaheuristic based on the r2 indicator for many-objective optimization. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 679–686. ACM, 2015.
- [45] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [46] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.
- [47] Anil Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31:651–666, 06 2010.
- [48] Anil K Jain and Richard C Dubes. Algorithms for clustering data. *Englewood Cliffs: Prentice Hall*, 1988, 1988.
- [49] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.
- [50] George Karypis and Vipin Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. <http://www.cs.umn.edu/~metis>, 2009.
- [51] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.
- [52] Marco Laumanns, Günter Rudolph, and Hans-Paul Schwefel. *Approximating the pareto set: Concepts, diversity issues, and performance assessment*. Citeseer, 1999.
- [53] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282, 2002.
- [54] Martin HC Law, Alexander P Topchy, and Anil K Jain. Multiobjective data clustering. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2004.
- [55] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 48(1):13, 2015.
- [56] Kaiwen Li, Rui Wang, Tao Zhang, and Hisao Ishibuchi. Evolutionary many-objective optimization: a comparative study of the state-of-the-art. *IEEE Access*, 2018.
- [57] Ke Li, Kalyanmoy Deb, Qingfu Zhang, and Sam Kwong. An evolutionary many-objective optimization algorithm based on dominance and decomposition. *IEEE Trans. Evolutionary Computation*, 19(5):694–716, 2015.

- [58] Ke Li, Qingfu Zhang, Sam Kwong, Miqing Li, and Ran Wang. Stable matching-based selection in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 18(6):909–923, 2013.
- [59] Miqing Li, Shengxiang Yang, and Xiaohui Liu. Shift-based density estimation for pareto-based algorithms in many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):348–365, 2013.
- [60] Miqing Li, Shengxiang Yang, and Xiaohui Liu. Pareto or non-pareto: Bi-criterion evolution in multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 20(5):645–665, 2016.
- [61] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at <http://cs.gmu.edu/~sean/book/metaheuristics/>.
- [62] Ailong Ma, Yanfei Zhong, and Liangpei Zhang. Adaptive multiobjective memetic fuzzy clustering algorithm for remote sensing imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 53(8):4202–4217, 2015.
- [63] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.
- [64] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11):205, 2017.
- [65] Anirban Mukhopadhyay, Ujjwal Maulik, and Sanghamitra Bandyopadhyay. A survey of multiobjective evolutionary clustering. *ACM Computing Surveys (CSUR)*, 47(4):61, 2015.
- [66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [67] Sriparna Saha and Sanghamitra Bandyopadhyay. A symmetry based multiobjective clustering technique for automatic evolution of clusters. *Pattern recognition*, 43(3):738–751, 2010.
- [68] Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.
- [69] Ryoji Tanabe, Hisao Ishibuchi, and Akira Oyama. Benchmarking multi-and many-objective evolutionary algorithms under two optimization scenarios. *IEEE Access*, 5:19597–19619, 2017.
- [70] Alexander Topchy, Anil K Jain, and William Punch. A mixture model for clustering ensembles. In *Proceedings of the 2004 SIAM international conference on data mining*, pages 379–390. SIAM, 2004.
- [71] Cor J. Veenman, Marcel J. T. Reinders, Eric Backer, and Eric Backer. A maximum variance cluster algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(9):1273–1280, September 2002.

- [72] Handing Wang, Licheng Jiao, and Xin Yao. Two_arch2: An improved two-archive algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 19(4):524–541, 2015.
- [73] Rui Wang, Robin C Purshouse, and Peter J Fleming. Preference-inspired coevolutionary algorithms for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 17(4):474–494, 2013.
- [74] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.
- [75] Andrzej P Wierzbicki. The use of reference objectives in multiobjective optimization. In *Multiple criteria decision making theory and application*, pages 468–486. Springer, 1980.
- [76] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.
- [77] Shengxiang Yang, Miqing Li, Xiaohui Liu, and Jinhua Zheng. A grid-based evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 17(5):721–736, 2013.
- [78] Eng-Juh Yeoh, Mary E Ross, Sheila A Shurtleff, W Kent Williams, Divyen Patel, Rami Mahfouz, Fred G Behm, Susana C Raimondi, Mary V Relling, Anami Patel, et al. Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling. *Cancer cell*, 1(2):133–143, 2002.
- [79] Yuan Yuan, Hua Xu, Bo Wang, and Xin Yao. A new dominance relation-based evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 20(1):16–37, 2016.
- [80] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.
- [81] Qingfu Zhang, Wudong Liu, and Hui Li. The performance of a new version of moea/d on cec09 unconstrained mop test instances. In *2009 IEEE congress on evolutionary computation*, pages 203–208. IEEE, 2009.
- [82] Xingyi Zhang, Ye Tian, and Yaochu Jin. A knee point-driven evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 19(6):761–776, 2015.
- [83] Yu-Hui Zhang, Yue-Jiao Gong, Jun Zhang, and Ying-biao Ling. A hybrid evolutionary algorithm with dual populations for many-objective optimization. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 1610–1617. IEEE, 2016.
- [84] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.
- [85] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 862–876. Springer, 2007.

- [86] Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *International Conference on Parallel Problem Solving from Nature*, pages 832–842. Springer, 2004.
- [87] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, 103, 2001.
- [88] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer, 1998.
- [89] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.