EVOLUTIONARY ALGORITHMS IN CLUSTERING: CHALLENGING PROBLEM GENERATION AND SEARCH SPACE ADAPTATION

A thesis submitted to The University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2020

Cameron William Benjamin Shand

Department of Computer Science

Contents

Word Count: 55784

List of Tables

List of Figures

List of Algorithms

Notation

N	Number of data points
D	Number of dimensions
K	Number of clusters
$\mathbf{x}$	A D-dimensional vector $\mathbf{x} = [x_1, x_2, \ldots, x_D]$
X	A dataset of N data points, i.e. $X = (\mathbf{x}_i)_{i=1}^N$
C_k	The kth cluster
$\lvert C_k \rvert$	The size (cardinality) of the kth cluster
$\mathcal{C}$	The set of (K) clusters, i.e. $\mathcal{C} = \{C_1, \ldots, C_K\}$
$d(\mathbf{x}_i, \mathbf{x}_j)$	Distance between data points $\mathbf{x}_i$ and $\mathbf{x}_j$
$\mathcal{I}_D$	D-dimensional identity matrix
$\mathcal{N}(\mu, \sigma^2)$	Normal/Gaussian distribution with mean μ and variance σ^2
$\mathcal{U}(0, 1)$	Uniform distribution between 0 and 1
$\mathrm{Dir}(\alpha)$	Dirichlet distribution
P_f	Stochastic ranking fitness probability parameter
$\boldsymbol{\mu}^{(k)}$	The kth cluster mean in HAWKS
$\boldsymbol{\Sigma}^k$	The kth cluster covariance matrix in HAWKS
$\widetilde{\boldsymbol{\Sigma}}^k$	The kth cluster diagonal covariance matrix in HAWKS

$\boldsymbol{R}^{(k)}$	The kth cluster rotation matrix in HAWKS
$\boldsymbol{S}^{(k)}$	The kth cluster scaling matrix in HAWKS
λ^{ratio}	HAWKS' eccentricity constraint (ratio of maximum and minimum eigenvalues)
overlap	HAWKS' overlap constraint
s_{all}	Silhouette width
s_{target}	Target silhouette width
$\bar{\boldsymbol{\mu}}$	The global centroid of all data points
$n_{\mathbf{x}}(1)$	The 1$^{\text{st}}$ nearest neighbour of data point $\mathbf{x}$
$n_{\mathbf{x}_i}^{-1}(\mathbf{x}_j)$	The rank of data point $\mathbf{x}_j$ in data point $\mathbf{x}_i$'s nearest neighbours
α_w	The winning algorithm in HAWKS' versus mode
α_l	The losing algorithm in HAWKS' versus mode
$\mathcal{P}$	Population of individuals
Γ	Set of relevant edges that form Δ-MOCK's reduced genotype
$\lvert\Gamma\rvert$	The length of Δ-MOCK's reduced genotype
δ	Percentage of relevant edges in the MST to keep for Δ-MOCK's reduced genotype
$sr\eta$	Calculation of δ as a function of N i.e. $\lvert\Gamma\rvert = \eta\sqrt{N}$
L	Δ-MOCK's neighbourhood size parameter
T	Number of resolution levels
G_{max}	Number of generations

Abstract

Identifying clusters in data has a wide range of applications, yet remains a surprisingly difficult task. The subjective nature of clustering leads to difficulty in both selecting the most appropriate algorithm for a given problem and evaluating the performance of these algorithms (as typically there is no ground truth to aid evaluation). Synthetic data can help us to understand the capabilities of algorithms, but only if this data is itself well-understood and presents challenges that are reflective of real-world clustering problems.

Current benchmark sets for clustering are limited in this regard. To address this issue, we propose a synthetic data generator (named HAWKS) for clustering that uses an evolutionary algorithm to optimize different challenges. We compare HAWKS against other popular generators and datasets for clustering. We find that HAWKS is able to both produce datasets that result in a wide spread of performance for clustering algorithms, and that the performance varies differently for different algorithms. We extend this generator to directly maximize the performance difference between two clustering algorithms, automatically finding their relative weaknesses without explicit parameterization of cluster properties. This can provide greater mechanistic insights and aid in algorithmic development.

In the last part of the thesis, we again explore the use of evolutionary algorithms in clustering, but for the assignment of data points to clusters using multiple objectives. We extend the Δ-MOCK algorithm to adapt the search space (which scales with the size of the dataset) in order to reduce computation and focus the search. By adapting the search space using the current performance and employing strategies to explore this space, at least equivalent performance is achieved for a near two-thirds reduction in computation time (compared to Δ-MOCK). We then use HAWKS to obtain greater insights into the relative differences in our proposed strategies by producing datasets with properties that were better suited to previously poor strategies.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Lay Summary

The ability to identify patterns within data, particularly identifying sub-groups (or *clusters*) that contain things that are more similar to each other than they are to other clusters, has a wide range of applications. For example, identifying patients that are more similar to each other can help with diagnoses, or finding similar images can help with automatic categorization.

To *accurately* identify this similarity however, we need to quantify it, so that we have an idea of *how* similar they are. Qualitatively, an image of an apple is more similar to that of a pear than to a pineapple, but all three are more similar compared to an image of a sheep. Quantifying the similarity is difficult — do we compare the colour or shape of the object in the image? What if instead we compare the composition or some other feature of the images? Not only can we measure similarity in different ways, but in order to *verify* whether these clusters are good (i.e. the images are indeed similar), we have many further possible choices. For example, should all images in a cluster be similar to each other, or just similar to a few images, which in turn are similar to a few images (and so on). Algorithms that detect these clusters are typically limited to a single perspective, meaning that they only work for certain types of data (which we do not know beforehand!).

To address all this subjectivity in constructing and verifying the clusters, we need synthetic data where we know *exactly* which clusters the objects belong to, allowing us to test whether the approach is suitable at detecting clusters. Creating synthetic data that mirrors the complexities of the real-world can be difficult however, particularly when doing it in a way that provides control over not just the overall difficulty, but different types of difficulty (caused by the different considerations of similarity).

In this thesis, we try to improve our ability to detect clusters in two ways.

In both approaches, we use a type of algorithm inspired by the principles of evolution to try and optimize multiple (conflicting) goals simultaneously. First, we propose a method of generating synthetic data where we try to optimize multiple types of difficulty, such that we can find out the capabilities and limits of cluster detection for different algorithms, so that for future real-world datasets (where we have no knowledge of what the clusters are) we can use more appropriate algorithms. Second, we try to optimize the detection of clusters by optimizing different measures of cluster similarity simultaneously, and try to improve the ability of this algorithm to deal with large amounts of data. To gain further insights into how this has affected the algorithm, we use our generator to tease out more nuanced differences.

Acknowledgements

To my supervisors, John Keane and Richard Allmendinger, I thank you for the many opportunities, laughs, advice, insights, and support you have given me. Thank you for taking a chance on someone who knew nothing of computer science. I hope this thesis proves it was worth it. Special thanks to Julia Handl who gave me many insights, advice, proofreads, and support, on nothing but pure academic interest. This thesis is significantly better due to your help. Thank you also to my examiners, Manuel López-Ibáñez and Richard Everson, both for their incredible attention to detail that has made this thesis much better, and for a surprisingly enjoyable viva that will inspire further work.

To avoid sounding more popular than I am, I will omit (most) names in the following. I trust you know who you are. Thank you to all the people who have enriched my time in Manchester. From the CDT cohort, to Gavin Brown and his amusing, illustrious (and surprisingly Greek) group, to the other members of MLO, to my board game group (who have to deal with my gleeful gloating), you've all made the last few years a joyous experience. Special thanks to Andrew Webb for always answering my questions, and showing me a new meaning to *mathematics*. Thank you also to my friends "down south" for always meeting me in London train stations, and keeping me distracted in the evenings.

To my sister, I hope this thesis casts a small shadow (just kidding). To my brother-in-law: good luck (not kidding). To both of you, thank you for being there for me and understanding that birthday presents can be given at arbitrary times. To my parents, it is difficult to articulate my appreciation for everything you have done for me. Your kindness, self-sacrifice, and outlook on the world embody (to me) the wonderful qualities of humanity. I hope that whenever I do grow up, I can do the same (step one is learning to drive...). To Georgiana, without your help and insight, this thesis would be noticeably worse. Without your love and support, my life would be immeasurably worse.

Chapter 1

Introduction

1.1 Is there a *best* clustering algorithm?

To answer this question, we must first define what clustering (also known as *cluster analysis*) is. Unfortunately, most definitions fall short of capturing the full spectrum of both the aims of clustering or even what the clusters themselves actually are (Ackerman, Ben-David, and Loker 2010; Hennig 2015; Jain, Murty, and Flynn 1999; Luxburg, Williamson, and Guyon 2012). In Chapter 2, an overview of these discussions is provided, but for our purposes here we can summarize clustering as trying to extract patterns from data or, more precisely, trying to find which groups of objects (such as different patients in medical data) are similar to each other, and dissimilar to other groups.

A frequent issue with clustering is the subjective nature of these clusters, and the many different definitions of 'similar' that can lead to valid or even contrary discoveries of clusters within data. The consequent existence of many clustering algorithms, each with their own nuanced assumptions, makes the selection of the *best* or *right* algorithm a non-trivial task. With the No-Free-Lunch (NFL) theorem for learning (Wolpert and Macready 1997) and non-universal definition of a cluster, there is no *best* algorithm for all datasets. For different types of datasets/problems (referred to as a problem class) however, a subset of algorithms may be superior to others. Attempts at validating the output of these algorithms (using cluster validation indices) is further complicated as these indices are often correlated with or used within the algorithms themselves, adding another layer of assumptions and potential bias to untangle. For each possible measure of cluster quality a clustering algorithm can be derived, creating a difficult cycle. Even

using many different algorithms (at great computational expense) may not be helpful if these results cannot be reliably compared. For an unbiased comparison, certain techniques can be used if there is a *ground truth*, such as the confirmed diagnosis for a patient, but as clustering is an *unsupervised* task this truth is typically absent i.e. we may only be able to identify patients who are similar, without definitive outcomes or diagnoses.

As different measures of cluster validity provide different perspectives, the natural development is to try to simultaneously use multiple such measures within a clustering algorithm. One popular approach in this regard is *evolutionary clustering*, which uses evolutionary algorithms as a method to tackle the resulting multi-objective optimization problem. In Chapter 7 we study and extend one such algorithm, Δ-MOCK. Although this algorithm simultaneously optimizes two complementary measures, this does not guarantee that the full spectrum of possible clusters can be captured. Arbelaitz et al. (2013) highlighted the issue of selecting complementary indices by studying 30 cluster validity indices, finding that no single index was always the most informative and that different indices are more useful for different types of dataset (which did not always correlate).

To illustrate the behaviour of different clustering algorithms, in Figure 1.1 (inspired by *scikit-learn*[1]), we can see the output of a number of clustering algorithms on different synthetic data.[2] This figure shows an array of different challenges that may be faced during clustering, and the resulting clusters that these algorithms discover (separated by colour). It is clear that on these datasets, some algorithms are generally better than others, but it is not easy to determine *a priori* which approach would be the best. Thus, it follows to ask how could the right algorithm be selected without exhaustive testing of many algorithms (and the additional complexity of tuning its hyperparameters).

[1] https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods

[2] These algorithms will be explained in Section 2.4, aside from Δ-MOCK which is the subject of Chapter 7. The synthetic data is generated using different functions and transformations in *scikit-learn*, aside from "*UKC*5" which is a real-world dataset discussed in Section 6.2.2 (for illustrative purposes we use a subset of this data) and HAWKS, which is the generator we propose in Chapters 5 and 6.

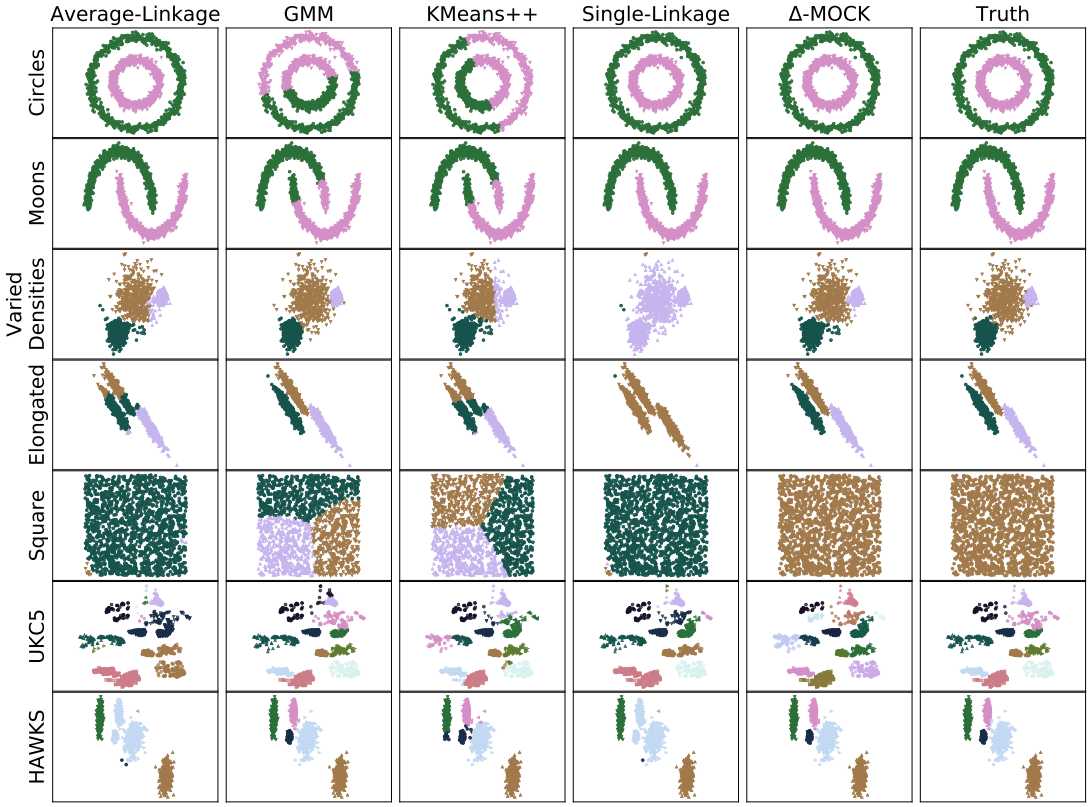


Figure 1.1: Illustration of clustering algorithms (inspired by *scikit-learn*[1] [Pedregosa et al. 2011])

1.2 Selecting the *right* clustering algorithm

As which can be considered the most appropriate clustering algorithm varies based on the dataset being studied, therein poses the question: through some kind of analysis on the dataset, can we select an algorithm (or subset of algorithms) that would be most appropriate? This underpins the algorithm selection problem (ASP) proposed in Rice (1976), where the aim is to predict which algorithm would be the best for a given dataset according to a set of problem features (quantitative measures of the dataset itself). This problem is tackled in different fields, such as *meta-learning* in the machine learning community or *exploratory landscape analysis* in the metaheuristics community (Mersmann et al. 2011; Smith-Miles 2008; Vanschoren 2019).

Addressing this problem, this thesis deals with: adapting the concept of problem features for cluster analysis; generating datasets that exhibit particular problem features with a view of creating a diverse test suite; and, generating datasets that are challenging for a particular algorithm (with a view to understand mechanistic differences between pairs of algorithms). These are further discussed in the remainder of this section.

A subsequent body of research has expanded upon Rice's framework for the ASP (Muñoz et al. 2018; Smith-Miles and Tan 2012; Smith-Miles et al. 2014; Smith-Miles and Bowly 2015), but a key part of this is the identification of the problem features, which is domain-specific. In clustering, we have the aforementioned issue of subjectivity when it comes to measuring what a cluster is, and so trying to capture the different facets of difficulty that a clustering algorithm face is complex.

Having a comprehensive set of descriptive problem features is only part of the problem; the existence of a number of datasets that have a wide range of diversity with respect to these problem features is necessary to obtain a deeper understanding of the advantages and disadvantages of any particular algorithm. Development of new/modified clustering algorithms would require analysis across a range of these different 'types' of difficulty in order to ascertain in what situations it is useful. At present, works that develop new clustering approaches often use a limited range of datasets, with no reflection upon the diversity of challenge among these datasets. The use of synthetic data is popular as it enables the use of *external* validation as the ground truth is available, though there are popular

real-world datasets that also have labels[3].

In order to generate synthetic data that has a diverse range of difficulty, we require a flexible generator. There have been many attempts at creating synthetic generators for clustering (Handl and Knowles 2005b; Iglesias et al. 2019; Qiu and Joe 2006a; Pei and Zaïane 2006; Pedregosa et al. 2011) with varying levels of complexity and capability. A primary focus of this thesis is the design of a synthetic data generator, named HAWKS[4], that optimizes (also using evolutionary algorithms to generate a population of datasets) the datasets themselves according to different measures that correspond to a particular type of difficulty.

The use of a synthetic generator to produce diverse datasets requires setting a range of parameters to produce these differences. If the desire is to compare the behaviour of clustering algorithms directly, it may not be known *a priori* which parameters are relevant to teasing out the differences between the algorithms. Therefore, directly producing datasets that maximize the difference in performance between these algorithms can facilitate the discovery of weaknesses, enabling deeper insights into the algorithms.

1.3 Thesis contributions

The main contributions of the thesis are stated more precisely as follows:

1. The development of a synthetic data generator that enables the parameterization of different cluster properties in order to modify the difficulty of the datasets (Chapter 5).

2. The comparison of existing synthetic data generators for clustering in terms of the diversity of these datasets, measured by performance across a range of clustering algorithms (Chapters 5 and 6).

3. The visualization of these datasets on an *instance space* using a set of informative features that describe different problem characteristics, in order to illustrate their diversity (Chapter 6 — Section 6.2).

[3]Such as the UCI machine learning repository (Dheeru and Karra Taniskidou 2017), though here classification datasets are often used for clustering, which can be misleading (Luxburg, Williamson, and Guyon 2012).

[4]Available at `https://github.com/sea-shunned/hawks`. For further software details, see Section A.1

4. The identification of areas in this instance space without datasets, and subsequent generation of datasets with properties to fill this area (Chapter 6 — Section 6.2).

5. The extension of said generator to produce datasets that directly maximize the performance difference between two clustering algorithms (Chapter 6 — Section 6.3).

6. The creation of a more flexible encoding for evolutionary clustering through the extension of an existing algorithm (Δ-MOCK) to reduce computation time and the importance of dataset-specific hyperparameterization (Chapter 7).

1.4 Thesis structure

Background on clustering is presented in Chapter 2, where we review different discussions and definitions for a cluster and cluster analysis as a whole; this is followed by discussions on key aspects of clustering (similarity measures, the impact of dimensionality) as well as different properties that a cluster can exhibit. We then discuss different types of clustering algorithms and the aforementioned cluster validation techniques to assess the quality of the output of these algorithms.

In Chapter 3 we present background on evolutionary algorithms, with an initial overview on the different components and the role they play in the optimization, followed by a focus on the importance and caveats when designing a suitable representation for the problem. We then look at methods for handling constraints in the optimization, and finally different methods for modifying the core hyperparameters of the evolutionary algorithm *during* the optimization.

Following this, Chapter 4 provides a more specific introduction to synthetic data generation. Therein lies a discussion of the need for synthetic data and the role it plays in empirical work and, in particular, the algorithm selection problem. Further discussion is given to synthetic data generators used to create datasets for clustering, as well as a body of work by Smith-Miles (Smith-Miles 2008; Smith-Miles and Lopes 2012; Smith-Miles and Bowly 2015; Muñoz et al. 2018) which extends the algorithm selection problem framework to evaluate the diversity of datasets, and thus the development of different methods to generate more diverse datasets.

After these background chapters, Chapter 5 will detail our first contribution in the form of HAWKS, a synthetic data generator that creates controllably complex datasets for more in-depth empirical analysis of clustering algorithms. The different components and design decisions of the generator will be explained (and tested where necessary), following a comparison with two other popular generators to evaluate the diversity of the datasets produced.

In Chapter 6 we further develop this generator, addressing some of the issues identified in Chapter 5. Namely, multiple mutation operators are compared in order to better converge in higher dimensions. We then use a wider set of parameters in HAWKS to generate more diverse datasets, comparing against a wider array of both synthetic and real-world data to evaluate diversity, which itself is measured using a wider pool of properties. We then extend HAWKS to more explicitly generate cluster structures that are difficult for particular algorithms. This is achieved by optimizing the performance difference between two algorithms in order to further understand the strengths and weaknesses of different approaches.

Our final contribution is presented in Chapter 7, where we discuss our improvement to a state-of-the-art evolutionary clustering algorithm, Δ-MOCK. Here, we integrate self-adaptation into Δ-MOCK to create a variant, Adaptive-MOCK, in order to address its biggest weakness: scalability. We then use datasets from HAWKS and our instance space from Chapter 6 to try and better ascertain the differences between our modifications to understand where Δ-MOCK and Adaptive-MOCK struggle to find the right cluster structure.

Finally, Chapter 8 summarizes the main conclusions of this thesis and outlines future research directions resulting from our work.

1.5 Publications

The work presented in this thesis has resulted in the following publications:

Shand, Cameron et al. (2018). "Towards an adaptive encoding for evolutionary data clustering". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. ACM, pp. 521–528. DOI: 10.1145/3205455.3205506.

Shand, Cameron et al. (2019). "Evolving controllably difficult datasets for clustering". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. ACM, pp. 463–471. DOI: 10.1145/3321707.3321761.

It should be noted that Chapter 5 is an expanded version of Shand et al. (2019), which was nominated for a Best Paper award at GECCO 2019, and Chapter 7 is an expanded version of Shand et al. (2018).

Chapter 2

Background: Cluster Analysis

This chapter introduces the fundamentals and challenges of cluster analysis to provide the necessary background for the work we explore in Chapters 5 and 6. We look at what cluster analysis is (Section 2.1) and the difficulty of defining what a cluster is (Section 2.1.2), the process of cluster analysis and the nature of similarity (Section 2.2), the properties we could consider for defining clusters (Section 2.3), the different types of clustering algorithms (Section 2.4), and finish with a discussion of how the quality of clusters can be quantified (Section 2.5).

2.1 What is cluster analysis?

The process of cluster analysis (synonymously referred to as *clustering*) is a natural one: given a set of objects, it is common to try and group together objects that are more similar to each other than they are to other (groups of) objects. These groups are generally referred to as *clusters*. For our purposes, these objects are data points. In this section, we provide a formal definition of clustering, followed by a discussion of the various arguments and definitions of what a cluster actually is, providing context to more broadly to define cluster analysis, thus laying the foundations before we both quantify and visualize different cluster properties later in this chapter.

2.1.1 Formal definition

In this thesis, we focus on a sub-domain of clustering where the input is a finite set of data points, and the output is a clustering (or partitioning) of those points.

A D-dimensional dataset, X, consists of N data points, i.e. $X = (\mathbf{x}_i)_{i=1}^{N}$. For a data point $\mathbf{x} \in X$, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$. A clustering, $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$ of X is a partitioning of X into K disjoint subsets, i.e. $\bigsqcup_{k=1}^{K} C_k = X$. The size of each cluster is represented by its cardinality, i.e. $|C_k|$ is the number of data points in the kth cluster.

A distance function (or metric), $d(\cdot, \cdot)$, defines a distance between two data points. By definition (Schweizer and Sklar 1960), a metric is:

1. Symmetric $d(\mathbf{x}_1, \mathbf{x}_2) = d(\mathbf{x}_2, \mathbf{x}_1)$
2. 0 when the data points are identical $d(\mathbf{x}_1, \mathbf{x}_1) = 0$
3. Adheres to the triangle inequality $d(\mathbf{x}_1, \mathbf{x}_2) \leq d(\mathbf{x}_1, \mathbf{x}_3) + d(\mathbf{x}_2, \mathbf{x}_3)$

where $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ are three arbitrary data points. As a result of these properties, distances are non-negative.

We then define a general clustering function that, given a dataset X and distance function d, outputs a partitioning (Ackerman, Ben-David, and Loker 2010). Throughout this thesis, clustering algorithms (irrespective of mechanistic differences) take this form of clustering function. Although we focus in this thesis on having individual data points between which we can calculate a distance, it is important to note that in general this is not a requirement for clustering, and that the input may be the *dissimilarities* between points which is then used to identify clusters, with or without knowledge of the original data points themselves.

2.1.2 What is a cluster, *really*?

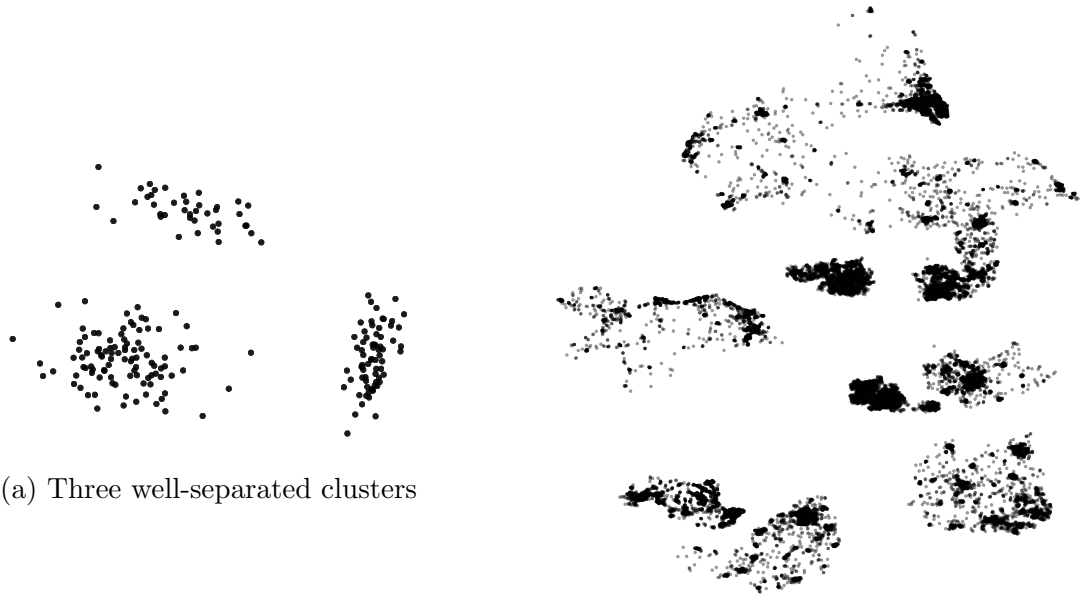
From our previous definition of clustering, a single cluster represents a group of similar points. Discussions of what 'similar' could mean aside (which we discuss further in Section 2.2.1), for some applications/data a cluster can represent a single concept (such as a group of patients with high blood pressure), yet this is still a loose and impracticable definition. How these clusters emerge from the data is itself debated; Jain (2010) argue that clusters are *natural* groupings of points inherent in the structure of the data, whereas Luxburg, Williamson, and Guyon (2012) argue that the clusters themselves are a direct consequence of the application. Hennig (2015) extends this argument philosophically to question the very nature of truth (natural vs. constructed) and the inherent limited observation capabilities of humans.

Using image clustering as an example, it is easy to understand how many different valid sets of clusters can be created depending on the purpose, in support of both interpretations above. For the same data, there can be different sets of clusters depending on the goal and thus notion of similarity. For example, a group of images may be clustered based on their chromatic similarity, or on the similarity of their contents (such as grouping together images of the same animal). Through this lens, these sets of clusters are both inherent to the data *and* to the application. Thus, irrespective of shape or other properties that we discuss later, navigating a multitude of arguments we can denote a cluster as a group of points that are similar, and the definition of this similarity is paramount to the clusters that are produced.

One clear point from this discussion is the need for real-world applications of clustering to have a defined purpose with a clear aim. The aim of cluster analysis supports both defining what the cluster is and how they should be quantified. For example, using clustering for exploratory data analysis aims to find patterns, where clusters represent a distinct statistical pattern. This in turn narrows the set of potential measures to quantify these clusters (Hennig 2015). It is important for the aim to be established *a priori*, in order to use appropriate measures for both the clustering itself and subsequent evaluation.

The subjectivity of what constitutes a cluster and thus the difficulty of this definition becomes clear when attempting to visually identify cluster membership. By way of example, Figure 2.1a shows three well-separated clusters (this should be unequivocal). When examining why these are distinct clusters, post-rationalizations will make reference to properties such as the clear separation or density of points, but humans have instinctual pattern recognition that does not consciously and explicitly evaluate such properties (Bowker and Star 2000; Everitt et al. 2011; Handl and Knowles 2007; Jain, Murty, and Flynn 1999).

This is exemplified in Figure 2.1b, which presents a more challenging problem such that our intuition begins to falter (and thus subjective opinions on cluster boundaries diverge). The data in this example are anonymized locations of crimes taken from UK Police (Garza-Fabre, Handl, and Knowles 2017). More conscious analysis is required to ascertain the cluster boundaries, and the unclear nature of the clusters results in ambiguity of the cluster membership for (importantly) the points on the borders of clusters. The actual clusters (as defined by the source) can be seen in Figure 2.2. This example illustrates that, even when easily



(a) Three well-separated clusters

(b) Complex real-world data

Figure 2.1: Examples of clustering datasets, with a simple three-cluster synthetic example (a), and a more complex real-world example (b).

visualizable, the multiple criteria and considerations of what constitutes a cluster become difficult for humans in spite of our innate ability at identifying them.

A lot of real-world data complicates this issue as it extends beyond visualizable dimensions, limiting the ability of humans to evaluate both the data and any clusters produced by algorithms without potential loss of information (through e.g. projection). As such, verification of clusters produced by clustering algorithms relies on some quantifiable assessment and/or domain expert evaluation (if available). In Section 2.5 we discuss different approaches to making such assessments.

2.1.3 Defining cluster analysis

There has been research that has attempted to better define the task of cluster analysis. These axiomatic approaches to formalizing clustering have tried to address the ill-defined nature of the problem, and as a result highlighted the impossibility of an all-encompassing clustering approach, showing the inherent trade-off in clustering as a task. Kleinberg (2002) formulated the *impossibility theorem*, showing that no clustering function (a function that takes the data as input and outputs a partitioning of this data) was able to satisfy three simple

Figure 2.2: The same real-world example (as Figure 2.1b) with the actual structure shown.

properties:

1. *Scale invariance* — scaling of the similarity function must not change the partitioning.

2. *Richness* — the clustering function must be able to output every possible partitioning.

3. *Consistency* — the output must be the same if a different distance function that shrinks intra-cluster and expands inter-cluster distances is used.

These properties are later extended in Ackerman, Ben-David, and Loker (2010) to create a wider set of properties that can be used to guide the choice of clustering approaches for a given task. This is achieved through the addition of other properties, such as *order invariance* (where the ordering of pairwise distances between points must remain under a different distance function), and the relaxation of previous properties, such as the use of inner and outer consistency to separate the consistency of intra- and inter-cluster distances.

Hennig (2015) lists potential characteristics of clusters that may be desirable in order to better identify what it is that cluster analysis approaches are trying to

define, but even within these characteristics there are multiple perspectives (such as the desire that "between-cluster dissimilarities should be large" and the myriad of ways to measure this may not completely agree). Ultimately, however, there is no unique formal definition of clustering (Adolfsson, Ackerman, and Brownstein 2019), though this may not be problematic, as evidenced by the utility and many applications of clustering as a method for data analysis.

We can look at common approaches to clustering to better identify what is *broadly* understood to be involved. If we observe the output of clustering algorithms, each data point is assigned to a cluster (akin to the clustering functions above) and therefore produces a partitioning. There are different types of output, however. For *hard* or *crisp* clustering, each data point is assigned membership to a single cluster, though in *fuzzy* or *probabilistic* clustering a data point can belong to multiple clusters (with different weights or probabilities). In this thesis, we focus on hard clustering.

2.2 Measuring clusters

In the previous section we discussed some different perspectives on what constitutes a cluster and the problem of clustering as a whole. In this section, we will look both more quantitatively and practically (through visualization of different cluster properties) to better understand cluster analysis.

Moving away from the formal definition of cluster analysis, we can instead look at the general steps or components, offering a practical perspective to the task. For this, Handl, Knowles, and Kell (2005) describe three main steps of cluster analysis, which are a condensed version of that outlined in Jain, Murty, and Flynn (1999):

1. *Pre-processing*: this includes feature selection (to make the similarity between features clearer), dimensionality reduction (integrated with feature selection, but more specifically to avoid the *curse of dimensionality*, which we discuss below), and the selection of a similarity measure.

2. *Clustering*: the actual selection of both the clustering algorithm and appropriate hyperparameters. This will be discussed in Section 2.4.

3. *Validation*: the evaluation of the resulting set of clusters, which can then feedback into the *pre-processing* and/or *clustering* stages. This will be

discussed in Section 2.5.

2.2.1 Pre-processing

The aim of this step is to better facilitate the discovery of clusters further downstream. This is achieved through methods such as feature selection (to reduce the dimensionality, remove uninformative or noisy features etc.) or feature extraction (combining features together through projection/embedding).

A key reason to reduce dimensionality is to make the similarity of data points clearer through avoiding the *curse of dimensionality*. In this phenomenon, the distances between disparate data points become increasingly similar as the dimensionality increases, so as $D \to \infty$ all data points become equivalent nearest neighbours (Bellman 1966; Beyer et al. 1999). This results in distance functions losing their efficacy at identifying clusters in higher dimensions. Kriegel, Kröger, and Zimek (2009) and Houle et al. (2010) provide further insights to the different causes and effects of this "curse", namely the distinctions between the number of *relevant* features, and the local relevance of these features (distinct subsets of the features may be relevant for subsets of the data, reducing the efficacy of global feature reduction methods). As a result, although higher dimensionality is likely to increase the difficulty in clearly identifying clusters (due both to an increasing similarity in distances between points, and the number of irrelevant features), it is dependent upon the data and not a universal truth (Fränti and Sieranoja 2018).

A famous dimensionality reduction method is principal component analysis (PCA), which projects the data onto orthogonal dimensions that maximize the variance (Dunteman 1989). Although useful and with wide prevalence, this approach is not well-suited for non-linear cluster structures or for outliers (Huber 1985). PCA has also been used and recommended as a quick method of identifying whether there is cluster structure within the data (Varmuza and Filzmoser 2009). More recent approaches such as t-SNE (Maaten and Hinton 2008) and UMAP (McInnes and Healy 2018) may be more suitable in reducing the dimensionality of more complex data while preserving the structure, which is paramount in order to identify clusters that are present in the data, and not an artefact of the reduction technique.

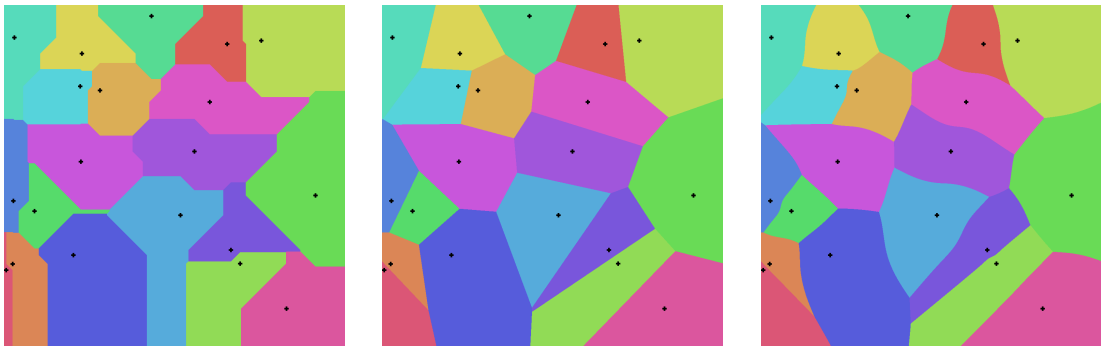
Measuring similarity

It is important to identify a measure of similarity, as it must be selected such that it allows for the identification of clusters within the data. Beyond obvious selections based on the type of data (e.g. numerical/categorical), the choice of similarity measure is a difficult yet vital task. The selection may be informed by intended algorithm, e.g. the use of Euclidean distance with K-Means (Jain 2010). The use of distance metrics (as defined in Section 2.1.1) is intuitive and natural for many applications of clustering (Jain, Murty, and Flynn 1999).

We further explain the importance of similarity measures by way of analogy. Generally, human settlements may be grouped into neighbourhoods (clusters). Only in sparsely populated, rural environments will such neighbourhoods be easily distinct, as opposed to urban environments where most neighbourhoods are interconnected over varying population/building densities. The decision on where to draw the distinction between separate neighbourhoods represents the distance threshold for cluster delineation i.e. a minimum degree of similarity. A natural way to measure the similarity between objects is the distance between them, of which the most common is the Euclidean distance. We could instead consider a conceptual similarity of neighbourhoods based on their features (landscape, building type, fauna etc.), but here we consider distance. We expect that objects (buildings) in the same neighbourhood to be much closer to each other than to objects in different neighbourhoods, but this typically occurs over a gradient (representing gradual, rather than step, changes in building density). Ignoring considerations about where in these neighbourhoods to measure the distance from, if we look at the distance measure itself there are further decisions: is the distance a straight point-to-point, or is it measured based on travel distance (which itself can vary by the mode)? As a consequence of such subtleties, there are numerous such measures (Anderberg 1973; Everitt et al. 2011; Jain, Murty, and Flynn 1999).

A common distance metric known as the Minkowski distance is defined for two data points, $\mathbf{x}$ and $\mathbf{x}'$, as follows:

$$d(\mathbf{x}, \mathbf{x}') = \left(\sum_{i=1}^{D} |x_i - x'_i|^q \right)^{1/q} \tag{2.1}$$



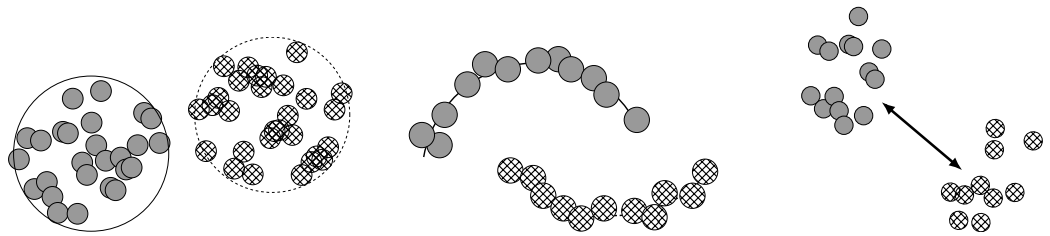
(a) $q = 1$ (b) $q = 2$ (c) $q = 3$

Figure 2.3: Voronoi diagrams of the Minkowski distance with different values for the order of the norm (q).

where q is a configurable parameter, such that setting $q = 1$ results in the Manhattan distance and $q = 2$ results in the Euclidean distance. The effect that q can have is shown in Figure 2.3, which shows a Voronoi diagram for $q \in \{1, 2, 3\}$. The coloured cell around a marked point identifies the area that is closest to that point according to that similarity measure (and thus a potential cluster boundary). After an initial area that is undoubtedly closest to a particular point, we can see that even different parameters for the same similarity measure result in quite different neighbourhoods/possible boundaries for a cluster around that point. Other distance functions may affect the boundaries more significantly, highlighting the importance of the distance function for discovering cluster structure.

Not all similarity functions are distance-based, however. A commonly-used similarity measure is the *cosine similarity*, which measures the angle between two vectors. However, some similarity measures (including the cosine similarity, which violates the triangle inequality) are not proper metrics (Korenius, Laurikkala, and Juhola 2007; Schweizer and Sklar 1960) and thus may not be wholly suited for clustering. Despite this, cosine similarity has seen extensive use in the field of information retrieval (Han, Kamber, and Pei 2011). Regardless of which type of similarity function is used, it can then be incorporated into the clustering algorithm to determine cluster membership. As we previously discussed, however, the choice of similarity measure should align with the intended aim of the cluster analysis and the intended similarity between objects in a cluster.

Having identified which points belong to the same cluster according to their respective similarity, a complete partitioning is created. The quality of this partitioning can then be validated (Section 2.5), the results of which can then be used



(a) Compactness (b) Connectedness (c) Separation

Figure 2.4: Cluster property categories (reproduced from Handl, Knowles, and Kell [2005]).

to feedback and adjust the feature selection/extraction methods, or algorithmic parameters.

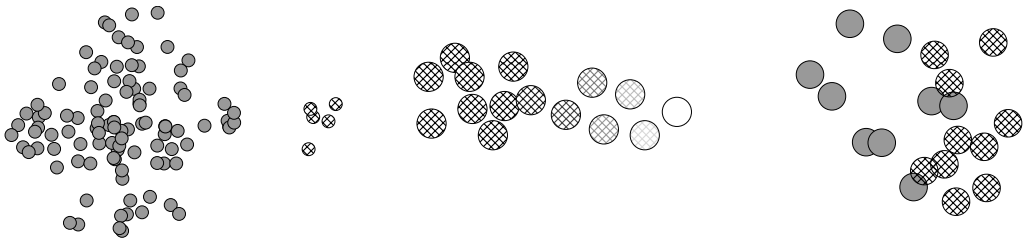
2.3 Cluster properties

In this section we explore different properties that can define a cluster, which can in turn help formulate criteria that can be used within clustering algorithms, or when assessing the *validity* or *quality* of clusters (Section 2.5).

Generally, these properties (and their various mathematical formulations) form the objective that clustering algorithms seek to optimize, and as such these properties can be used to categorize both the measures (also referred to as "clustering criteria") and algorithms. Handl, Knowles, and Kell (2005) identified a broad categorization of these properties, illustrated in Figure 2.4: *compactness*, *connectedness*, and *spatial separation*.

Compact clusters contain members that are all close to one another and therefore also close to a global representative of these points (such as the centroid). As a result, these clusters are generally (hyper-)spherical and algorithms that use these measures (such as K-Means, which we outline in the next section) are less effective when the cluster shapes are more complex.

Connectedness is a property that looks at the proximity of points to their nearest neighbour(s), rather than all members of a cluster simultaneously. This can enable detection of non-convex cluster structures, but can be susceptible to outliers or when there is little separation between clusters. Density-based algorithms fall into this category as density demands close proximity between nearest neighbours (Ankerst et al. 1999), as well as algorithms that look at clusters with respect to distances between a subset of points, such as single-linkage (Sibson



(a) Size/density (b) Outliers (c) Overlap

Figure 2.5: Challenging cluster properties

1973). In Figure 2.4b, some objects that are clearly part of the same cluster are further from each other than they are to objects in the other cluster, highlighting the "chaining" effect of these connected points. Generally, clusters that are quite elongated (or eccentric) would be highly connected, but not compact.

In the original categorization, *spatial separation* is noted to be a useful property in the presence of other criteria, such as a measure of cluster sizes or densities, as by itself it can be a naïve measure. Whereas compactness looks at *intra-cluster* similarity between all objects in a cluster, the spatial separation looks at *inter-cluster* similarity. Although the compactness and connectedness differ in the resolution at which they look (with compactness considering the cluster as a whole and connectedness looking at a local neighbourhood), the resolution of spatial separation may vary depending on how this separation is measured. The inter-cluster separation may be measured using neighbouring clusters (e.g. looking at the distance between the closest two clusters) or the average distance between a cluster and all others. As such, these measures can range from effectively measuring the distance between the two closest data points from different clusters, to the average distance between *all* data points of different clusters. Clearly, this property is more loosely defined that the compactness and connectedness, hence the original authors noting that it "is usually combined with other objectives" (Handl, Knowles, and Kell 2005).

These three categories can be put further into context when we consider other related properties, illustrated in Figure 2.5. Large differences in cluster densities can cause problems for clustering algorithms, such as the scenario illustrated in Figure 2.5a where it may be favourable (in terms of the criteria used by the algorithm) to consider the small cluster as part of the larger one. The density of clusters (or lack thereof) can play an important role in identifying cluster

boundaries; the two clusters shown are both compact, yet could be challenging to separately identify using a compactness-based measure or algorithm. Similarly, clusters of vastly different sizes can result in smaller clusters being numerically irrelevant during the optimization when using measures that average across objects (such as the silhouette width which we discuss later in Section 2.5.1).

Similarly, outliers can be difficult to detect as it may not be clear at which point objects no longer belong to the cluster and are therefore considered to be outliers (Figure 2.5b). For algorithms that primarily consider connectedness, this can be identified through e.g. a distance threshold that defines the neighbourhood around objects to limit the connectedness, such that disconnected points are outliers. Appropriate parameterization of this threshold is paramount to performance, but without knowledge of the true cluster membership it is a significant challenge.

A lack of separation can result in an overlap of clusters (Figure 2.5c), which can result in the underlying cluster structure being irrevocably lost. As we explore later in Section 5.1.3, there are different ways of identifying when points overlap, but visually at least this is easy to identify. Data that exhibits high levels of overlap cannot be clustered, but robustness to low amounts of overlap (where a small number of points from different clusters are in close proximity) varies between algorithms and is a challenge that needs consideration for complex data.

As previously discussed, the multitude of properties that could define similarity and therefore a cluster make enumeration of potential properties difficult. The properties discussed in this section are intuitive, however, and used extensively in the literature. They are therefore useful to consider later in this thesis when we try to analyze the expected difficulty of datasets based on its clusters.

2.4 Clustering approaches

Referring to the second step of cluster analysis in Section 2.2, we need an actual clustering algorithm itself to assign cluster membership to the data points. There have been various categorizations of clustering algorithms, depending on whether the representation of the clusters is being categorized e.g. hierarchical against partitional (Jain, Murty, and Flynn 1999), or the approach is categorized between e.g. “combinatorial” (looking at the observed data only) vs. model-based (Hastie, Tibshirani, and Friedman 2009).

The literature for clustering algorithms is vast, and is covered well in various texts (Duda, Hart, and Stork 2001; Everitt et al. 2011; Han, Kamber, and Pei 2011; Hastie, Tibshirani, and Friedman 2009; Jain, Murty, and Flynn 1999; Jain 2010). We describe a few commonly-used clustering algorithms that we will use in Chapters 5 and 6, as an understanding of these algorithms is needed to interpret the results in those sections, as well as some other popular algorithms for context.

2.4.1 Partitional clustering

Partitional clustering algorithms create a *single* partitioning, optimizing a particular clustering criteria (Jain, Murty, and Flynn 1999). As a single set of clusters is produced, a common issue with this category of approaches is that the number of clusters (K) is typically required as input to the algorithm. K-Means is a popular algorithm in this category that suffers from the need to set K *a priori*, when it is often unknown. In Section 2.5.2 we discuss several methods for trying to estimate K with these approaches, but there is no definitive method for this.

It should be noted that "K-Means", a term introduced in MacQueen (1967), more generally refers to the process of trying to partition a group of data into K sets, and there are multiple algorithms for this. The main variants are: 'Lloyd' (Lloyd 1982), 'MacQueen' (MacQueen 1967), and 'Hartigan-Wong' (Hartigan and Wong 1979). The nuanced differences between these are beyond the scope of this thesis; further details can be found in Morissette and Chartier (2013). Throughout this thesis, we generally refer to this approach as the K-Means algorithm for simplicity.

At its core, there are two main steps to K-Means: an *assignment* and *update* step. As noted by Bishop (2007), these two steps are synonymous with the expectation and maximization steps of the expectation-maximization (EM) algorithm (which we will further discuss later). Following the generation of an initial set of K cluster seeds (or, K "means"), which in the original formulation of this approach are randomly created, in the first step each data point is *assigned* to its closest seed. The second step then *updates* the initial seeds to become centroids (the average point in space of the assigned data points) to include the newly assigned data points. These two steps are repeated, iteratively updating the assignment of data points to the closet centroid and then updating the centroids themselves, for either a pre-defined number of iterations or until there is no further change in cluster assignment. Both of these steps serve to find the partition

that (locally) minimizes the objective function for K-Means, which is defined as:

$$\arg\min_{\mathcal{C}} \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2, \tag{2.2}$$

where $\mathcal{C} = \{C_1, \ldots, C_K\}$ is the set of K clusters, and $\|\mathbf{x} - \boldsymbol{\mu}_k\|^2$ is the (Euclidean[1]) distance between the data point $\mathbf{x}$ and the centroid $\boldsymbol{\mu}_k$ of the kth cluster (Han, Kamber, and Pei 2011). The greedy nature of K-Means results in it converging to local optima, and is therefore dependent on the initial location of the seeds. To alleviate this, Arthur and Vassilvitskii (2007) introduced K-Means++, where the initialization scheme ensures a wider distribution of the initial seeds across the input space to avoid situations such as dense clusters starting with two (and thus splitting the cluster). This is achieved by iteratively selecting seeds with a weighted probability such that the data points which are furthest from existing seeds are selected. Another popular strategy is to attempt multiple random initializations and select the best assignment (in terms of minimizing Equation 2.2), though this can also be used with K-Means++.

Owing to the initialization issue, the need to specify K, and the bias of the intra-cluster variance towards compactness, K-Means is applicable to a narrow range of cluster structures. However, the commonality of these structures alongside the computational speed (and ease of implementation) is such that the algorithm has been widely used (Everitt et al. 2011; Han, Kamber, and Pei 2011; Jain, Murty, and Flynn 1999).

Gaussian mixture models (GMMs) are another popular approach to clustering that we use later in this thesis. GMMs provide (in the context of clustering) a probabilistic way to model clusters via a combination of Gaussian distributions. We write the mixture distribution of K Gaussians as:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{2.3}$$

where $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, and π_k are the means, covariances, and mixing coefficients for the K distributions. The EM algorithm (Dempster, Laird, and Rubin 1977) is an iterative method for finding the maximum likelihood, which in the case of GMM is used to optimize these three parameters (means, covariances, and mixing

[1] K-Means can use other distance measures, but its canonical definition uses the Euclidean distance (Jain 2010).

coefficients). As with K-Means, the initialization of the parameters influences both the final results and the computation required to converge (to a local, not global, maximum). As GMM requires more computation than K-Means, these parameters are typically initialized using a single run of K-Means. Note that this is the case with the implementation available in Pedregosa et al. (2011) that is used in this thesis. For further details on initialization methods for GMMs, see Melnykov and Melnykov (2012).

As noted by Bishop (2007), we can directly relate GMM to K-Means by fixing each covariance matrix to $\epsilon\mathcal{I}$, where ϵ is a fixed constant and $\mathcal{I}$ is the identity matrix. As $\epsilon \to 0$, maximizing the log likelihood is equivalent to the minimization of Equation 2.2 for K-Means, and we obtain a hard assignment of cluster membership.

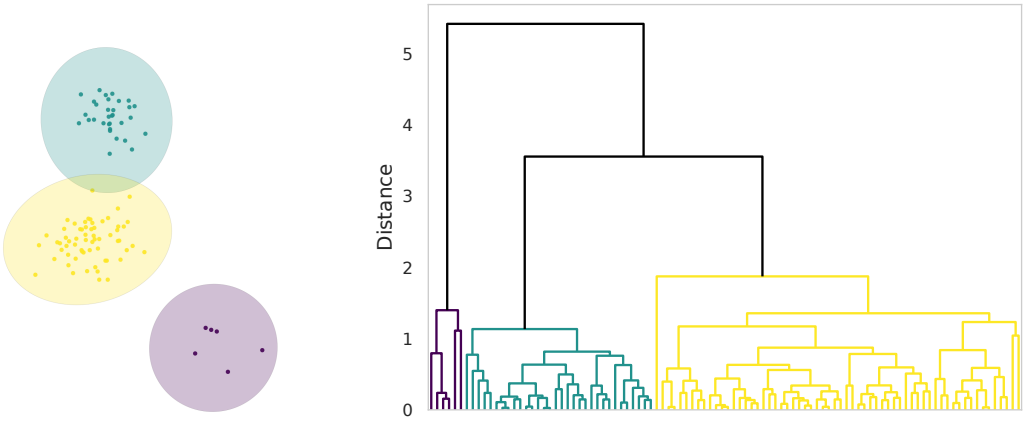
There have been several other extensions/modifications to K-Means, from modifications to the initialization (such as the aforementioned K-Means++) to modifications of the distance function used such as kernel K-Means (where the use of a kernel permits extraction of cluster that are non-linearly separable), and K-Medoids (where data points are used as the centroids, limiting the set of possible centroids). The latter is particularly useful when using other distance functions Bishop (2007).

Other popular partitional clustering algorithms include spectral clustering, which uses the eigenvalues of the similarity matrix to then partition the data, often outperforming K-Means (Luxburg 2007). DBSCAN (Ester et al. 1996) is a density-based algorithm that categorizes points based on how many other points are within a pre-specified distance, which advantageously enables handling of noise and mixed densities, though is highly sensitive to the definition of the neighbourhood (Kriegel et al. 2011).

2.4.2 Hierarchical clustering

A primary use of clustering is to explore the structure of data, and as such not only is the "optimal" K unknown, but analyzing cluster structure across a range of K could prove beneficial. A common example is a biological taxonomy, where different resolutions of clusters may wish to be explored without loss of meaning (e.g. species can be members of different clusters, yet at the genus-level they belong to the same cluster; Ward Jr [1963]).

Hierarchical clustering methods enable this by constructing a *dendrogram* that



(a) Example dataset (generated by HAWKS)

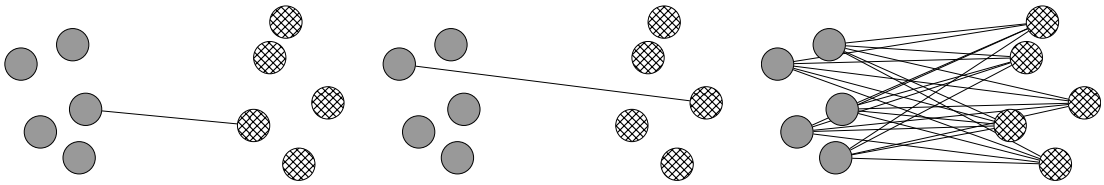
(b) Full dendrogram

Figure 2.6: Example dataset (a) and its accompanying dendrogram, using average-linkage (b).

groups the data at different levels which correspond to a different number of clusters. This dendrogram can be constructed in full (ranging K from 1 to N) or to a specified K only. Figure 2.6 shows an example dataset and corresponding dendrogram, where we can see the different splits corresponding to the cluster structure shown.

Hierarchical clustering algorithms construct the dendrogram using one of two approaches: bottom-up/*agglomerative* or top-down/*divisive* (Rokach and Maimon 2005). In the former, each data point begins in a cluster by itself and the clusters are iteratively merged until every point is in a single cluster. In the latter, all data points begin in a single cluster and are iteratively split until each cluster is a single data point.

Clusters are either merged or split using a criterion (which varies between algorithms), which itself uses a similarity measure. There are three canonical linkage algorithms that utilize a different criterion, illustrated in Figure 2.7. Single-linkage considers the distance between two clusters as the *shortest* distance from any member of one cluster to any member of another cluster (or, more generally, the two most similar members). Conversely, complete-linkage defines the distance between two clusters as the *maximum* distance of any member of one cluster to another (Jain, Murty, and Flynn 1999; Kaufman and Rousseeuw 1990). Average-linkage takes the average distance between every member of both clusters. These three methods are illustrated in Figure 2.7. At each step, a single merge (or split)



(a) Single-linkage (b) Complete-linkage (c) Average-linkage

Figure 2.7: Illustration of three methods for selecting which two clusters to merge in agglomerative hierarchical clustering. The lines indicate which distance(s) are considered as *the* distance between the two clusters.

of the two identified clusters is performed until the desired number of clusters is obtained.

Typically, agglomerative clustering is favoured over divisive due to its simplicity when choosing which pair to merge or split on (Edwards and Cavalli-Sforza 1965), although this computation is repeated many more times as typically $K << N$, i.e. the number of clusters is much lower than the number of data points (Duda, Hart, and Stork 2001; Walter et al. 2008). For a lower K, with the added motivation of the inherent greedy nature of hierarchical clustering, divisive methods can produce better clusters (Guénoche, Hansen, and Jaumard 1991; Macnaughton-Smith et al. 1964).

2.5 Cluster validation

The final step of cluster analysis (as outlined in Section 2.2) is validating the output of a clustering algorithm. The validation (or assessment) of a given partition is paramount to determining the performance of the algorithm, but the inherently unsupervised and subjective nature of clustering makes this difficult. This section discusses some of the major approaches used for cluster validation, namely: cluster quality/validity measures, evaluation of the "correct" number of clusters, and statistical tests for cluster structure.

2.5.1 Cluster validity indices

To assess the quality or validity of a cluster, we require explicit mathematical formulation of one (or more) cluster properties (Section 2.3) in order to provide a quantitative answer to the question "how good are these clusters"? Each measure, or *cluster validity index*, formulates one or more of the properties in different ways,

providing different answers to this question. Luxburg, Williamson, and Guyon (2012) note that "A unique, global, objective score for all clustering problems does not exist", summarizing both the difficulty of this task and motivating the need for using multiple such measures. There are two types of indices: *internal* and *external*. Commonly used methods for each type are discussed below.

Internal validity indices

Internal validity indices use information inherent to the data, such as the maximum distance between two points that have been assigned to the same cluster. As previously discussed, these measures each take into account a subset of cluster properties and thus cannot capture the full spectrum of cluster quality. The substantial number of indices prohibit their description here; for more information of these measures see Arbelaitz et al. (2013). Of particular note is that this study compared 30 validity indices in multiple environments and concluded that no single index was clearly superior, and some were more useful for different scenarios (such as the amount of noise).

Silhouette width Owing to its importance in Chapter 5, the silhouette width (Rousseeuw 1987) is fully defined here. Conceptually, it considers aspects of intra-cluster compactness and inter-cluster separability. Silhouette width values are in a well-defined range, $[-1, 1]$, that is comparable across datasets of similar dimensionality. A value of 1 represents very compact and well-separated clusters, whereas a negative silhouette width value indicates that points in different clusters are not well-separated (and that their cluster membership should be changed). The silhouette width for a single data point, $\mathbf{x}_i$, is defined as:

$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max\{a(\mathbf{x}_i), b(\mathbf{x}_i)\}} \tag{2.4}$$

where

$$a(\mathbf{x}_i) = \frac{\sum_{\mathbf{x}_j \in C_k} d(\mathbf{x}_i, \mathbf{x}_j)}{|C_k|}; \qquad \mathbf{x}_i \neq \mathbf{x}_j;\ \mathbf{x}_i \in C_k;\ C_k \in \mathcal{C}, \tag{2.5}$$

$$b(\mathbf{x}_i) = \min_{C_k \in \mathcal{C}} \frac{\sum_{\mathbf{x}_j \in C_k} d(\mathbf{x}_i, \mathbf{x}_j)}{|C_k|}; \ \mathbf{x}_i \notin C_k \tag{2.6}$$

where $|C_k|$ is the kth cluster's cardinality and $d(\mathbf{x}_i, \mathbf{x}_j)$ is the distance between data points $\mathbf{x}_i$ and $\mathbf{x}_j$. Here, $a(\mathbf{x}_i)$ represents the cluster compactness (with respect to $\mathbf{x}_i$) and is the average distance from $\mathbf{x}_i$ to all other data points in its cluster. The second term, $b(\mathbf{x}_i)$, represents the separation between clusters; for data point $\mathbf{x}_i$ this is defined as the minimum of the average distances to all data points in every other cluster. The silhouette width is calculated for all N data points in dataset X, and an average value is taken to obtain the overall silhouette width:

$$s_{all} = \frac{1}{N} \sum_{\mathbf{x} \in X} s(\mathbf{x}_i). \tag{2.7}$$

Other notable cluster validity indices are the Davies-Bouldin index (Davies and Bouldin 1979), and the Dunn index (Dunn 1973). The Davies-Bouldin index provides an average similarity between each cluster and its most similar one, where lower values indicate a better clustering (note that while zero is the minimum, there is no defined maximum). The Dunn index provides a measure of the ratio between the minimum distance between points in different clusters compared to the largest distance within a cluster, thus directly incorporating more information about all clusters simultaneously.

These three (silhouette width, Davies-Bouldin index, and Dunn index) internal indices are measures that represent a ratio between cluster compactness and (inter-cluster) separation, with different formulations to calculate this ratio. As noted both by Bezdek and Pal (1995) and Bolshakova and Azuaje (2003), some components in these formulations can be modified to give different variants of these indices (the former work proposed a set of "generalized Dunn indices"). Their formulation as a ratio, however, provides some utility over measures that just consider the separation or compactness separately, yet expression as a singular value can result in information loss (Handl, Knowles, and Kell 2005).

The use of ratios is not required, however, as many popular indices consider only one of the three properties (compactness, connectedness, and separation). The aforementioned K-Means method uses the intra-cluster variance, a compactness-based measure, for which many other variants (taking the square root, using the medoid instead of centroid etc.) can be derived (Bezdek and Pal 1998). There are fewer properties for measuring the connectedness, though examples include the "connectivity" (Handl and Knowles 2007) and "k-nearest neighbour consistency" (Ding and He 2004) to capture the cluster assignment of

nearest neighbours. Measuring the separation shares similarities with the compactness, in that a few number of decisions (which data points from clusters should be used in the measure, the use of an average or maximum/minimum etc.) can combine to create many related measures. For a more complete enumeration and discussion of internal validity indices, see Arbelaitz et al. (2013).

External validity indices

External validity indices require a ground truth so that the cluster assignment can be evaluated directly. Generally, these indices produce a value that can be used to compare the validity of partitions from different algorithms and/or for different datasets.

As previously discussed, cluster labels are useful insofar as they represent the cluster structure for the intended application i.e. they are indubitable. Luxburg, Williamson, and Guyon (2012) argue that it is an *assumption* that the class labels coincide with cluster structure, which refers back to the example of clustering images producing different results based on the end-goal. The provided labels need to match the application in mind for these external indices to provide relevant information. For synthetic data, the labels correspond to the underlying generating mechanism and are thus sound. For the real-world data we use in this thesis, we assume that the labels are appropriate, and are therefore applicable to use with these external measures.

Meilă (2007) showed through analysis of the popular external validity indices that the presence of desirable properties (such as invariance to the number of data points and a bounded value comparable across datasets) varied and that, as with internal indices, an index cannot have all desirable properties and thus be universally applicable. Despite using the labels, external validity indices do not necessarily offer a completely unbiased or unequivocal measure of cluster quality due to e.g. assumptions in the null hypothesis (Meilă 2007), though steps (such as normalizing with the expected value on random data) can be taken to reduce bias (Handl, Knowles, and Kell 2005; Hubert and Arabie 1985).

Adjusted Rand Index (ARI) An adjustment of the popular Rand Index in Hubert and Arabie (1985) to create the Adjusted Rand Index (or ARI), corrected for chance such that random data results in a score of 0. This is an example of attempts to correct for some bias in these measures (yet still deficient as per

Table 2.1: Contingency table comparing two partitions $\mathcal{C}$ and $\mathcal{C}'$

		Partition $\mathcal{C}'$				
	$\mathcal{C} \backslash \mathcal{C}'$	C'_1	C'_2	$\dots$	C'_s	Sums
Partition $\mathcal{C}$	C_1	n_{11}	n_{12}	$\dots$	n_{1s}	n_{1+}
	C_2	n_{21}	n_{22}	$\dots$	n_{2s}	n_{2+}
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
	C_r	n_{r1}	n_{r2}	$\dots$	n_{rs}	n_{r+}
	Sums	n_{+1}	n_{+2}	$\dots$	n_{+s}	

Meilă [2007]), and as a result the ARI is used extensively (Santos and Embrechts 2009). The ARI has an upper bound of 1, where 0 is the expected value for random cluster assignment, and the maximum value of 1 for completely correct assignment (Steinley 2004). The lower bound is dependent upon the number clusters in the clusterings being compared (Chacón and Rastrojo 2020).

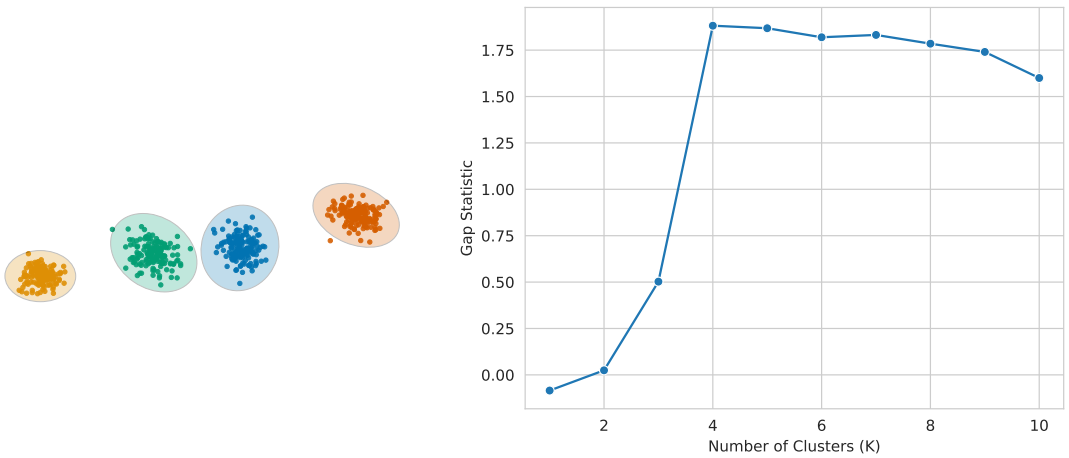
Owing to our extensive use of the ARI, we fully define it here. Table 2.1 shows a contingency table comparing two partitions, $\mathcal{C} = \{C_1, \dots, C_r\}$ and $\mathcal{C}' = \{C'_1, \dots, C'_s\}$, where r and s are the number of clusters for each partition. Each entry in the contingency table is a count of the number of data points in common between those clusters, i.e. $n_{ij} = |C_i \cap C'_j|$. The ith row and column sum of the contingency table are denoted by n_{i+} and n_{+i} respectively, i.e. $n_{i+} = \sum_{j=1}^{s} n_{ij}$.

The ARI is then calculated as follows:

$$\text{ARI} = \frac{\sum_{i=1}^{r}\sum_{j=1}^{s}\binom{n_{ij}}{2} - [\sum_{i=1}^{r}\binom{n_{i+}}{2}\sum_{j=1}^{s}\binom{n_{+j}}{2}]/\binom{N}{2}}{\frac{1}{2}[\sum_{i=1}^{r}\binom{n_{i+}}{2} + \sum_{j=1}^{s}\binom{n_{+j}}{2}] - [\sum_{i=1}^{r}\binom{n_{i+}}{2}\sum_{j=1}^{s}\binom{n_{+j}}{2}]/\binom{N}{2}} \tag{2.8}$$

where n_{ij}, n_{i+}, and n_{+j} are taken from the contingency table (Hubert and Arabie 1985). Although the ARI can measure the similarity between any two partitions, since it is typically used as an external validity index one of these partitions is the ground truth.

Measures from supervised machine learning, such as the F-measure (Rijsbergen 1979) or mutual information (Romano et al. 2014) can also be used as external validity indices. The use of these measures can provide a different view of the quality of the clustering rather than comparing clusters individually from two clusterings (Handl, Knowles, and Kell 2005).



(a) Example dataset (generated by HAWKS)

(b) Gap statistic for varying K

Figure 2.8: Example of the gap statistic, with the dataset (a) and accompanying graph (b) used to identify the "knee" that occurs at $K = 4$, indicating the likely true number of clusters.

2.5.2 How many clusters are there?

As previously discussed, many clustering algorithms require the number of clusters as an input parameter, whereas typically this is an unknown quantity that can vary by preference or application (Everitt et al. 2011; Tibshirani, Walther, and Hastie 2001). The necessity for this is exacerbated when, as is the case with algorithms such as K-Means, their own objective functions are optimized when there is one cluster per data point. As a result, using external methods and/or information to select the appropriate number of clusters is needed.

One method of discovering appropriate values of K is to generate multiple partitions across a range of K and then compare the performance using an aforementioned validity index, or the *gap statistic* (Tibshirani, Walther, and Hastie 2001). The "gap" measured is the difference between the within-cluster sum of squares for the data and a random reference distribution. The use of a compactness measure does bias this method towards compact cluster structures, limiting its utility (as a gap may not be observed for any K). For data with well-defined structure, however, the optimal K should create a bend (referred to as a "knee" or "elbow") in the graph as separated clusters are initially grouped into a single cluster with an underestimated K, and split internally into multiple clusters with an overestimated K.

Figure 2.8 shows a dataset and the associated gap statistic values for varying K, where the peak occurring at $k = 4$ gives a (correct) indication of the true number of clusters. This approach is useful for finding the most appropriate K value, which is essential for clustering algorithms such as K-Means that rely on it. The gap statistic can also be used as a framework: in Bayá and Granitto (2013), the authors used a similarity measure based on the minimum spanning tree (MST[2]) instead of the within-cluster sum of squares to identify "arbitrary-shaped" clusters, as opposed to purely compact clusters.

Chiang and Mirkin (2010) review multiple different methods that can be used to estimate the number of clusters, which include the use of the gap statistic and aforementioned silhouette width. The datasets used in their experiments, however, are simple synthetic Gaussian data which do not sufficiently examine a broad enough range of cluster structure to fully compare the differences between these approaches, limiting the conclusions of their work.

Some clustering approaches can find for themselves an optimal (or range of good) K values implicitly, without the need for defined measures to estimate it. In Chapter 7, we introduce *evolutionary clustering*, a group of methods that use evolutionary algorithms (Chapter 3) to discover the cluster structure. Owing to their population-based nature, these methods have the potential to implicitly generate a population of solutions of different K values, allowing further analysis or examination by a domain expert to select a single partitioning.

2.5.3 Testing for cluster structure

Most internal methods of validation assume that there is cluster structure to begin with, which can be unknown and difficult to ascertain. The adjustment for the ARI helps assign a score of 0 to randomly assigned points, but not all measures have this desirable property. Figure 1.1 showed empirically that clustering algorithms can identify clusters where none exist, which can be particularly pernicious when interpreting results (Handl, Knowles, and Kell 2005; McShane et al. 2002). Thus, statistical tests can be useful to ascertain whether there is structure in the underlying data and therefore if clustering is likely to produce clusters of meaning (Adolfsson, Ackerman, and Brownstein 2019). Their use is restricted, however, as Everitt et al. (2011) note that the statistical power of these tests can

[2]The minimum spanning tree is the subset of edges that connect all nodes (without cycles) such that the sum of the weights (distance between points in this case) is minimized.

be limited. The inherent assumptions of the particular null hypothesis being used limit the cluster structures that can be detected.

The aforementioned *gap statistic* does implicitly test for structure, as a random reference distribution is used as a baseline (Tibshirani, Walther, and Hastie 2001). If the data has no structure (under the same assumptions as the null reference distribution, which typically is a simple uniform distribution over the range of observed values), then no gap is observed for any K (Everitt et al. 2011). Without some *a priori* knowledge of the underlying data distribution, misleading results can be obtained.

Adolfsson, Ackerman, and Brownstein (2019) performed a study of *clusterability* methods, using both real-world and carefully simulated data to tease out differences between the approaches. The methods were analyzed through the lens of three requirements: *efficiency*, *algorithm independence*, and *effectiveness*. The first requires that the method be computable in polynomial time, such that they can be reasonably applied to large datasets. The second requirement is that the approach should not be specific to a particular algorithm, as it would not be able to identify cluster structure that the algorithm itself could not identify. The third requirement is that the method itself is effective; that, at the very least, the method is able to identify the clusterability of datasets for which it is clear-cut (such as the lack of structure in data drawn from a single Gaussian or clear structure of two well-separated Gaussians). This work identified clear differences between methods, importantly laying the foundations for further work which has hitherto somewhat lacked study (even though it is an important step in the *pre-processing* step of cluster analysis, as outlined in Section 2.2).

2.6 Summary

This chapter provided an overview of cluster analysis, highlighting the different properties that can be considered when defining and (with difficulty) quantifying exactly what constitutes a cluster. Some common clustering algorithms were introduced, followed by a discussion of the different methods that can be used to validate the resulting clusters from these algorithms, both in terms of their structure and hyperparameters such as the appropriate number of clusters.

Chapter 3

Background: Evolutionary Optimization

This chapter outlines single- and multi-objective optimization, describes the general components of a genetic algorithm (GA), and covers fields relevant to the work described later. This includes discussion of the considerations for problem representation in Section 3.2.1 and how this interacts with the other components of a GA. A discussion of the different types of constraints and how to deal with them is given in Section 3.3, which is an important foundation to Chapter 5. Finally, a discussion of parameter control and tuning is given in Section 3.4, which plays an important role in Chapter 7.

3.1 Single- and multi-objective optimization

Optimization is the process of finding a solution ($\mathbf{x}$) from a set of solutions ($\mathcal{X}$) that is the 'best' with respect to some criterion formalized as an objective function (f). Formally, we can define a general optimization problem as:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{X}, \\ & g_i(\mathbf{x}) \leq 0, \ i = 1, \ldots, m \\ & h_j(\mathbf{x}) = 0, \ j = 1, \ldots, p \end{aligned} \tag{3.1}$$

where g_i are the m inequality constraints and h_j are the p equality constraints, which constrain the possible values that the solution ($\mathbf{x}$) can have. When $m =$

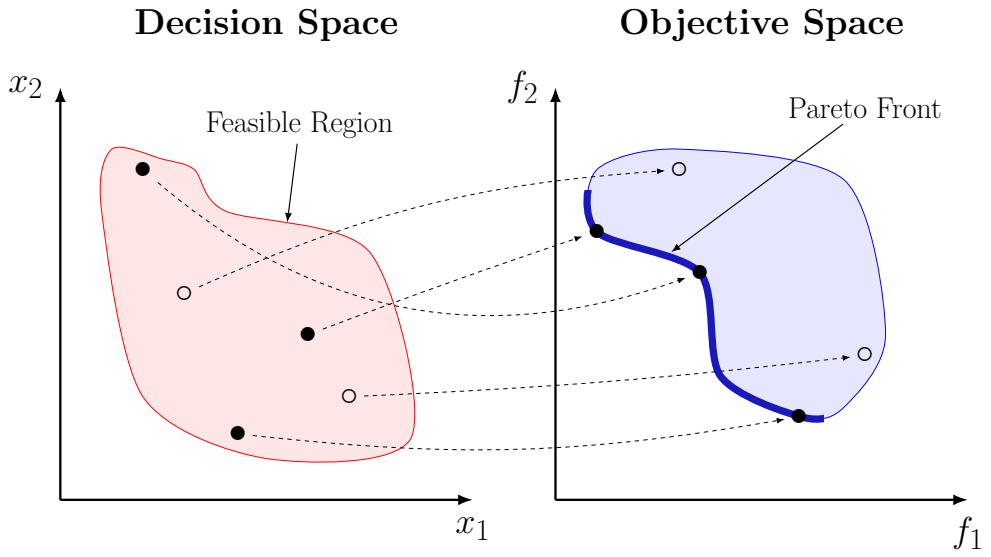


Figure 3.1: Mapping of non-dominated solutions (•) and dominated solutions (○) from the decision space to the 2-dimensional objective space, with the Pareto front is highlighted by the thicker line (▬).

$p = 0$, it is an unconstrained optimization problem. A solution, $\mathbf{x}$, consists of l decision variables i.e. $\mathbf{x} = [x_1, \ldots, x_l]$. The feasible search space, $\mathcal{X}$, is the set of solutions, where any solution within this space satisfies the constraints and is the set over which the search is performed. Note that here we have defined a minimization problem, though this extends to maximization with no loss of generality (as minimizing $f(\mathbf{x})$ is equivalent to maximizing $-f(\mathbf{x})$). Our objective function, $f(\mathbf{x})$, provides a mapping ($f : \mathcal{X} \mapsto \mathbb{R}$) from our solution/decision space to our objective space.

Often, there are multiple criteria or objectives that we desire to be simultaneously optimized, creating a multi-objective optimization problem (MOOP; [Emmerich and Deutz 2018]). Here, the definition for an n-objective problem is similar to the single-objective version shown in Equation 3.1:

$$\begin{aligned} \text{minimize} \quad & (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})) \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{X}, \\ & g_i(\mathbf{x}) \leq 0, \ i = 1, \ldots, m \\ & h_j(\mathbf{x}) = 0, \ j = 1, \ldots, p \end{aligned} \tag{3.2}$$

where we now have a vector of n objectives that are to be minimized, and a mapping $f : \mathcal{X} \mapsto \mathbb{R}^n$ to an n-dimensional objective space. Figure 3.1 shows such

a two-dimensional objective space and the mapping of solutions.

3.1.1 Pareto optimality

For the comparison of solutions with multiple objectives, we use the concept of Pareto-dominance. For two feasible solutions, $\mathbf{x}_1$ and $\mathbf{x}_2$, $\mathbf{x}_1$ Pareto-dominates $\mathbf{x}_2$ (denoted $\mathbf{x}_1 \succ \mathbf{x}_2$) if $\mathbf{x}_1$ is at least equal to $\mathbf{x}_2$ in every objective, and better in at least one objective (Zitzler et al. 2003). If neither solution dominates the other, then they are *non-dominated* solutions. Formally, we define Pareto-dominance (for a minimization problem) as:

$$\begin{aligned} \mathbf{x}_1 \succ \mathbf{x}_2 \iff & f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2)\ \forall i \in \{1, \ldots, n\} \\ & \wedge \\ & \exists\, i \in \{1, \ldots, n\} \colon f_i(\mathbf{x}_1) < f_i(\mathbf{x}_2). \end{aligned} \tag{3.3}$$

A more relaxed version of this is *weak* Pareto-dominance, where $\mathbf{x}_1$ weakly dominates $\mathbf{x}_2$ (denoted $\mathbf{x}_1 \succeq \mathbf{x}_2$) if $\mathbf{x}_1$ is at least equal to $\mathbf{x}_2$ in every objective, i.e. $\mathbf{x}_1 \succeq \mathbf{x}_2 \iff f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2)\ \forall i \in \{1, \ldots, n\}$.

If there is no solution that dominates $\mathbf{x}_1$, then $\mathbf{x}_1$ is *Pareto-optimal* i.e. $\nexists\, \mathbf{x}_2 \in \mathcal{X} : \mathbf{x}_2 \succ \mathbf{x}_1$. The complete set of Pareto-optimal or non-dominated solutions, $\mathcal{X}'$, form the *Pareto-optimal set* in the decision space, which maps to the *Pareto front* in the objective space, thereby representing the full spectrum of trade-offs between the objectives. Note that it is not always possible to generate the full Pareto front (Zitzler et al. 2003). In Figure 3.1 we show three non-dominated solutions across the full Pareto front, alongside the mapping of these solutions and the two dominated solutions from the decision space to the objective space. Generally in MOOPs the goal of an optimizer is to find an approximation of the Pareto front, for which quantifying the quality of the front is a non-trivial task (see Section 3.2.3 for a more in-depth discussion).

3.1.2 Solving optimization problems

Optimization algorithms used for these problems come in many forms and can be categorized in many ways, depending on whether they are: gradient-based (such as gradient descent) or gradient-free (such as the Nelder-Mead method [Nelder and Mead 1965]); deterministic (hill-climbing) or stochastic (genetic algorithms

— see Section 3.2); and, population-based (particle swarm optimization [Kennedy 2010]) or trajectory-based (simulated annealing [Kirkpatrick, Gelatt, and Vecchi 1983]). One categorization is of a family of approaches that includes the subject of this chapter: metaheuristics.

Metaheuristic methods are higher-level frameworks that guide the search process, aiming to efficiently explore the search space (but with no guarantee of finding optimal solutions), and are generally problem-independent but allow for the integration of domain-specific knowledge (Blum and Roli 2003). Thus they are suitable for problems where exact methods are not applicable/practical (Talbi 2009). Such problems occur when the landscape is either very complex (reducing the utility of gradient-based methods) or the search space is very large (e.g. the travelling salesman problem). When the objective cannot be expressed in analytical form, and the gradient is therefore unavailable, the problem is referred to as *black-box* optimization. Here, the objective function (the black-box) can be queried, but aside from the resulting output no other information is known about the function.

As highlighted in Chapter 1, this thesis deals with problems that require exploration of the trade-off between multiple, competing objectives. As there is no analytical function that can define a dataset (Chapter 5) or an exact method to cluster a dataset (Chapter 7), metaheuristic optimizers are highly effective in our work. In the next section, we motivate and describe such a group of optimizers: evolutionary algorithms.

3.2 Evolutionary algorithms (EAs)

Evolutionary algorithms (Goldberg 1989; Holland 1975), henceforth referred to as EAs, have been used extensively as a method of approximating the Pareto-optimal set for the optimization problems previously described. As EAs are population-based methods, they can be used to generate multiple approximately Pareto-optimal solutions that can then be used (for example) to gain further insights into the nature of the trade-off between objectives.

Various terminology is used when describing EAs, and the definitions can vary slightly between subcommunities. As such, Table 3.1 provides definitions used in this thesis to a number of standard terms; these terms will be used and explained further in their respective sections. It should be noted that there is a huge amount

of variety within EAs due to their modular and flexible nature, which allows for design choices to be made specific to a problem. Here, we describe commonly-seen components as informative examples.

EAs are based on concepts from evolutionary biology. In short, a population of individuals is generated and evolves over a number of generations where the "fitter" individuals are more likely to pass on their "traits" to their offspring, which undergo some random perturbation such that they are different from their parents. Over time, the fitness of the individuals generally rises as the traits of the fitter individuals are preserved and more are found.

Bäck, Fogel, and Michalewicz (2000) note that originally, EAs consisted of three main approaches: evolutionary programming (EP; Fogel, Owens, and Walsh [1966]), genetic algorithms (GAs; Holland [1962] and Holland [1975]), and evolution strategies (ESs; Rechenberg [1978] and Schwefel [1977]). These approaches were originally quite distinct, though in recent years the modular nature of these approaches has facilitated the use of methods from other areas, such that there is now some degree of overlap. An example of this is the use of adaptive mutation strategies within GAs, a method that originated in the ES community (Bäck 1992). This thesis is primarily concerned with GAs, though as we utilize concepts from related fields the concept of GAs being a subset of EAs is noted.

The basic outline of a GA can be seen in Figure 3.2. There, the One-Max problem is used to illustrate this example (Schaffer and Eshelman 1991). In this problem, each individual is a string of l bits i.e. each decision variable can take values $\{0, 1\}$. The fitness of an individual is simply the sum of the bits, which is to be maximized. A brief description of each step/component in this outline will be given in Section 3.2.2, but for context we first discuss how an individual represents a solution.

3.2.1 Representations

To successfully and efficiently use EAs, an appropriate or meaningful representation of the problem is required. The way that the solutions are encoded can greatly affect the ability of the algorithm to find the optimal solution(s), as well as heavily influencing the design of the genetic operators. This section outlines some of the fundamentals underpinning representations, and the importance of their design in the context of the algorithm as a whole. For a more complete overview, the reader is referred to Rothlauf (2006) and Rothlauf (2011).

Table 3.1: Definitions of terms in EAs (in alphabetical order).

Term	Definition
Allele	A single value in the chromosome. Synonymous with decision variable.
Child/Offspring	An individual created from a parent (or parents).
Chromosome	A single solution. Synonymous with individual.
Crossover/ Recombination	A genetic operator that swaps alleles between two or more individuals to create offspring.
Decision variable	A single value in the chromosome. Synonymous with allele.
Encoding	The method in which an individual represents the problem.
Environmental selection	Individuals from the combined pool of the parents and offspring are selected to form the population for the next generation.
Gene	A group of conceptually/phenotypically related alleles that can exist in some representations. Can refer to a single allele.
Generation	A single iteration of the algorithm.
Genetic operator	An operator that helps introduce genetic changes into the population to find individuals with higher fitness.
Genotype	The direct encoding/representation of an individual — this is what the genetic operators operate on.
Fitness	The quality of an individual.
Individual	A single solution. Synonymous with chromosome.
Mutation	A genetic operator that randomly perturbs an individual to try and introduce new genetic material.
Parental selection	One or more individuals are selected from the population to generate offspring.
Phenotype	The decoded solution, i.e. the actual solution to a problem that can be evaluated.

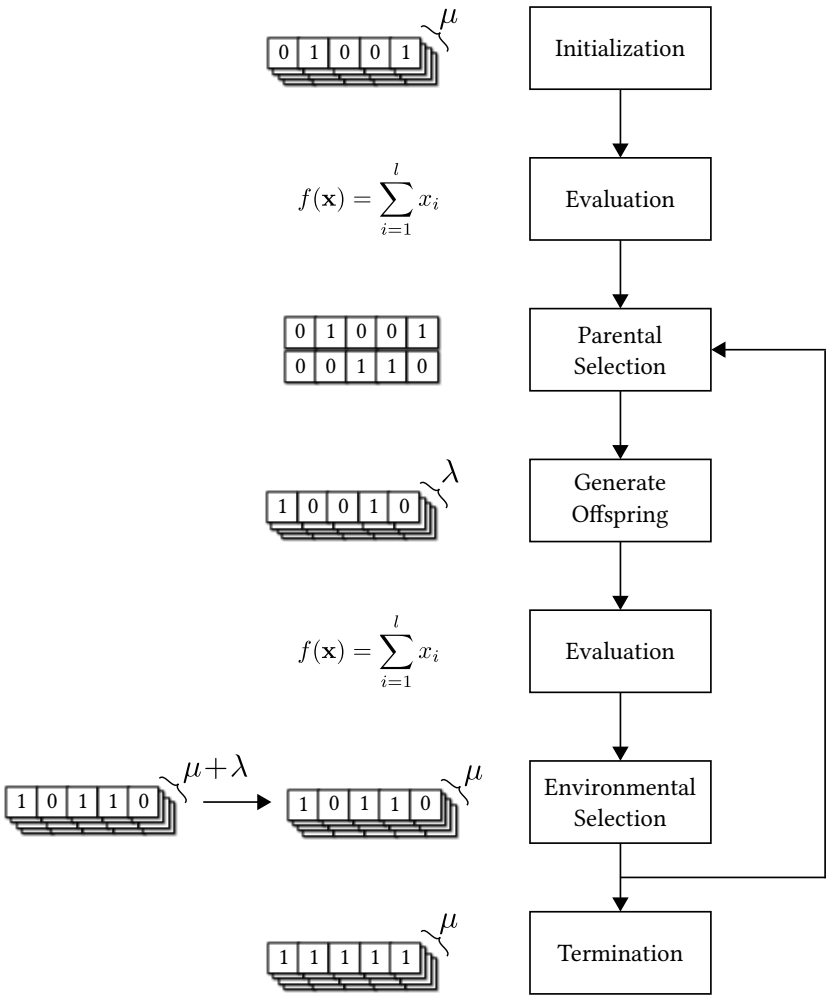


Figure 3.2: Outline of a standard genetic algorithm with a population size of μ and offspring size of λ applied to the classic One-Max problem.

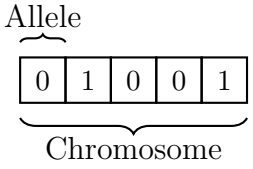


Figure 3.3: Example binary representation of $l = 5$ bits.

Figure 3.3 shows the representation for a single individual. The pictured chromosome contains all the "genetic" information for this individual. For our purposes, we refer to this string of variables synonymously as a chromosome/genotype/solution.

This genotype is the space in which the genetic operators modify the individual (rather than the phenotype), and how different individuals interact (e.g. through recombination). When these individuals are evaluated, then we refer to the *phenotype* of the individual. For example, if our genotype encodes a tree then its phenotype is the tree itself. The genotype and the phenotype can be the same (referred to as a *direct representation*), but they are conceptually distinct (Rothlauf 2006).

Types of Representation

A binary representation is illustrated in Figures 3.2 and 3.3. While the use of a binary representation simplifies the encoding, allowing for more intuitive design of genetic operators (e.g. bit-flip mutation), for some problems this is not a *natural* encoding as it may require integer or real-valued representations instead (Schraudolph and Belew 1992). These representations expand the possible search space for a given length, however. For a genotype of length l, a binary genotype can have 2^l possible states, which governs the size of the complete search space. For integer-based and real-valued representations the search space expands to $\mathbb{N}^l$ and $\mathbb{R}^l$ respectively, vastly expanding the number of possible states.

Despite the larger search space, similarity of the encoding to the problem itself can aid design (through using more specialized operators or initialization schemes) and reduce ambiguity of the genotype-phenotype mapping. Similarly, different representations can increase the number of possible values for a single allele, but overall reduce the length of the genotype which can affect the size of the search space differently. These considerations become important when designing a suitable representation.

Redundancy

A representation is redundant if there are more genotypes than phenotypes, leading to different genotypes having the same fitness. Rothlauf and Goldberg (2003)

make the further distinction between synonymously and non-synonymously redundant representations. In the former, genotypes that represent the same phenotype are in the same neighbourhood of the genotype space, i.e. they are few mutations apart. In the latter, genotypes that represent the same phenotype are not similar, which makes EAs more akin to random search as small modifications to fit individuals can result in drastically different phenotypes and therefore fitnesses.

Rothlauf (2006) notes mixed research with regards to the utility of redundant representations, as it seems problem-dependent whether redundancy and neutral mutations (a mutation that neither increases nor decreases fitness) quicken or slow exploration of the fitness landscape. This does not hold for non-synonymously redundant representations, however, as their low-locality (similar genotypes with dissimilar phenotypes) means that applying perturbations (via the genetic operators) to individuals with a higher fitness does not result in similarly high fitness individuals (Rothlauf and Goldberg 2003; Choi and Moon 2008). Understanding the redundancy is important for the representation, and is a consideration made for our design of representing a dataset in Section 5.1.1.

As previously discussed, the representation has a lot of importance to the design and potential performance of an EA. As a result, they tend to be particularly problem-specific, with limited general recommendations for their design (Goldberg 1989; Rothlauf 2006). Nonetheless, they are important in our work either through our design of a representation in Section 5.1.1 or in the extension of an encoding in Chapter 7 where working with only a subset of the genotype is paramount both to computational cost and performance.

3.2.2 GA components

In this section we discuss the major components of a GA as shown in Figure 3.2, which provides a broad outline for the One-Max problem. An initial population is created, and the fitness of each individual is evaluated. Parents are selected from the population to undergo crossover and mutation to generate offspring. The offspring are then evaluated and compete with their parents to be selected for the next generation. This continues until a pre-defined termination criterion (typically a number of generations) has been met. Further details of these steps are provided in the following sections.

Initialization

Initialization is the process of generating the initial population of μ individuals from which we can then vary and evolve. The method for initialization is highly dependent on the problem, and can greatly affect convergence. Simple methods for initialization involve generating random solutions or using a Latin hypercube (Branke 2012), which have no regard for fitness or feasibility but (importantly) should create diverse individuals.

Some knowledge of the problem can be embedded into the initialization scheme, however, to generate solutions that are (hopefully) closer to and/or more diverse along the Pareto front. Of course, this requires either knowledge about the problem or some intuition as to what may generate good solutions. The bias towards generating solutions with a particular subset of values may also reduce diversity, which needs consideration (Bennett, Xiao, and Armstrong 2004; X. Li et al. 2011). A successful example of a specialized scheme is seen in Garza-Fabre, Handl, and Knowles (2017), where the initialization produces individuals which are themselves an approximation of the Pareto front (this approach is discussed further in Section 7.1.1). A key consideration in the use of specialized initialization schemes is to ensure that there is still diversity in the initial population, thereby maintaining the utility of recombination and generally the diversity of genetic material. In the example above, the authors found that the initialization did not result in premature convergence or a lack of diversity, as is the risk with such protocols (Handl and Knowles 2007; Garza-Fabre, Handl, and Knowles 2017).

Evaluation

The fitness/objective function is evaluated for each individual and is used to compare and select individuals either as parents or which will survive to the next generation. While obvious that optimal solution(s) should be assigned the highest fitness, similar solutions should also have a high fitness to assist the guided search (a high-locality representation assists here). This can affect the choice or design of the objective function (if it is not determined by the problem). As the primary method of guidance for the search, the design of the fitness function is key for obtaining solutions of interest. In the consideration of this design, the representation (both its locality and decoding mechanism) and genetic operators need to be considered (Rothlauf 2011).

The computation of the fitness function can vary greatly. If the function

is available in closed form, then this may be straightforward, where the driver of computation time is dependent on whether the function scales on the length of the genotype or another input. When this scaling leads to prohibitive run times, methods such as limiting the precision of the function in the early stages and increasing as the run progresses can be useful (see Section 3.4.1 for further discussion).

Fitness functions that require simulations (Ong et al. 2005) or real experiments (Allmendinger and Knowles 2013), however, can greatly dominate the running time for the algorithm when days are required per evaluation. In these cases, surrogate-assisted methods can be preferred, where the fitness function is an approximation that is used to enable cheaper evaluations. Such an approach introduces a host of its own challenges — for more information see Allmendinger et al. (2017), Chugh et al. (2019a), Jin (2011), and Santana-Quintero, Montano, and Coello Coello (2010).

Parental Selection

Parental selection is concerned with selecting parents that will be used to create offspring. Common notation is the selection of λ parents, which may be different from the population size (μ). These methods generally utilize the fitness to bias the selection of more fit individuals to become parents, in the hope that variations of these fit individuals will lead to even higher fitness. We outline two commonly used methods: *fitness proportionate selection* and *tournament selection*.

Fitness proportionate selection directly uses the fitness of an individual as the probability for selection as a parent. This can be quantified using the normalized proportion of fitness that an individual has (as seen in Figure 3.4) or by ranking the individuals and calculating the probability based on the rank (Eiben and Smith 2015). The selection pressure that this method imposes, however, can fluctuate and deleteriously affect the search. This can occur at the beginning of the optimization (when exploration is preferred) where a strong selection bias towards fitter individuals can result in prematurely converging to local optima (Whitley 1989). Although somewhat alleviated by using rank-based selection over the absolute fitness, small differences in fitness in later generations can result in low selection pressure (Eiben and Smith 2015). For the sampling of individuals based on the probabilities calculated above, a roulette wheel can be used. As can be seen in Figure 3.4, the individual selected to be a parent depends on which

$f(\mathbf{x}_1) = 15 \ /100 = 0.15$

$f(\mathbf{x}_2) = 44 \ /100 = 0.44$

$f(\mathbf{x}_3) = 7 \ \ /100 = 0.07$

$f(\mathbf{x}_4) = 21 \ /100 = 0.21$

$f(\mathbf{x}_5) = 13 \ /100 = 0.13$

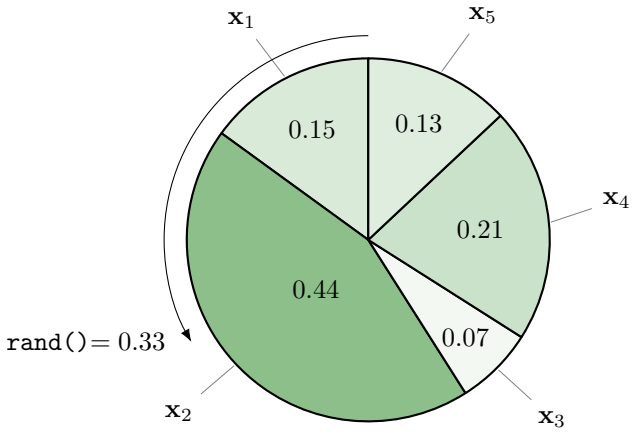


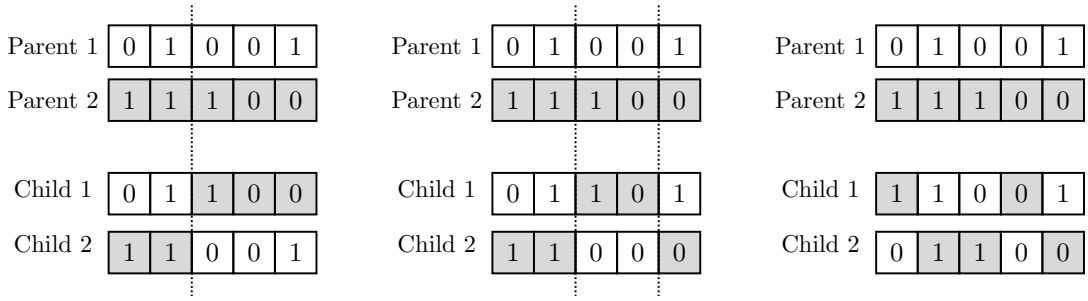
Figure 3.4: Illustration of fitness proportionate selection with roulette wheel sampling for a population of $\mu = 5$ individuals. Each individual occupies a range proportional to its relative fitness, and a random number is sampled to determine which individual is selected.

"bucket" the random number falls into.

Tournament selection, in contrast, does not simultaneously look at the entire population but a subset. A number (s) of random individuals are selected for the tournament, where the individual with the highest fitness "wins" and is selected to become a parent. The tournament size (s) determines the selection pressure of this method (Miller and Goldberg 1995). In the extreme, when the whole population is used in the tournament, the individual with the highest fitness is always selected. The individuals selected from the tournament can be with or without replacement, which can lead to the best solution having either on average or exactly s copies in the mating pool respectively (Rothlauf 2011; Sastry and Goldberg 2001). Typically, a binary ($s = 2$) tournament is used to ensure the selection pressure is not too elitist (Bäck 1995; Goldberg and Deb 1990). Whichever method is used for parental selection, it repeats until λ parents have been selected for the next step.

Genetic Operators

The genetic (or variation) operators modify the parents to create offspring, typically through *recombination* (crossover) and/or *mutation*. Crossover can be seen as allowing for the recombination of existing genetic material between parents, whereas mutation generally represents the ability to add genetic material into the population. Although Holland (1992) notes crossover as the main search operator



(a) One-point crossover (b) Two-point crossover (c) Uniform crossover

Figure 3.5: Illustrations of the three most common crossover operators.

for GAs, later work (Bäck 1992) showed the utility of mutation (dissuading the use of 'selectorecombinative' GAs, which omit mutation [Goldberg 2002]). The locality of the representation and degree of perturbation from the operators will determine the distance of the objective space that can be explored from a given individual.

Figure 3.5 shows three commonly used crossover operators: one-point, two-point, and uniform crossover (Holland 1975; Rothlauf 2011; Syswerda 1989). One-point crossover picks a random point in the chromosome, where all alleles after this point are swapped between the parents. Similarly, two-point crossover selects two points around which the swap will occur. Uniform crossover swaps each allele between the parents with a mixing ratio of 0.5. By taking (on average) half of the genetic material from either parent, offspring can be generated with less contiguous material from their parents compared to the n-point crossover methods. For a problem that has a particular sequence or permutation of a set (e.g. a combinatorial optimization problem), the chromosome may need a reparation step after crossover to ensure validity, a process which may be non-trivial (depending on the representation and problem).

As these crossover operators swap material between parents, they are thus independent from the type of representation (binary, continuous etc.). More specialized operators, such as the arithmetic or geometric crossover operators (Michalewicz and Schoenauer 1996) which attempt to mix or interpolate between parents, may be restricted to particular types. Regardless, Surry and Radcliffe (1996) provide some design principles for crossover operators to support the principle that the resulting offspring should not be more dissimilar to their parents than the parents are to each other (Radcliffe 1991). The probability used for crossover can vary, but typically values of 0.6–0.8 are used (Eiben and Smith

2015).

Mutation operators are generally more specific to the encoding (or problem), and as a result do not have 'standard' operators in the same way that crossover does. From simple operators such as bit-flip mutation (flipping a 0 to a 1 and vice versa) for binary encodings to specialized operators for covariance matrices (Section 5.1.4) or graph encodings (Section 7.1.1), the design can greatly affect the ability of the algorithm to explore the search space.

As mutation usually occurs at the allele-level (in contrast to the whole genotype in crossover), in general mutation probabilities for each allele are $\frac{1}{l}$ so that on average one mutation occurs per individual (Mühlenbein 1992). Typically, more focus is given to the tuning of both the probabilities and parameters for mutation than for crossover (see Section 3.4).

Environmental Selection

Environmental selection (also referred to as *survivor selection* in Eiben and Smith [2015]) is the process which selects the individuals from the pool of parents and offspring that will continue ("survive") to the next generation. Methods used for parental selection can be used here, but various methods have been proposed specifically for this task. Eiben and Smith (2015) categorizes these methods into age-based and fitness-based replacement. Environmental selection plays an important role in balancing some of the behavioural trade-offs of GAs, namely between exploration and exploitation, and between population diversity and selective pressure (Whitley 1989).

The simplest form of age-based environmental selection is "generational replacement", where the offspring are selected in their entirety for the next generation. Using different values for the population and offspring sizes (μ and λ, respectively), however, creates different variations of this method, such as generating a single offspring and replacing the current oldest individual (Jong and Sarma 1992).

Similar to parental selection, many methods use the fitness of the individual to decide whether it survives to the next generation. A popular example of this is elitism (De Jong 1975), where a number of the most fit individuals are selected for the next generation. These can either be selected from the offspring (Grefenstette 1986), or more commonly from the combined pool of parents and offspring (Eshelman 1990). The pressure that this exerts can be scaled by the

proportion of fit individuals carried over to the next generation (in the range $[1, \mu]$). The aforementioned issue of premature convergence can arise depending on the degree of selection pressure towards the most fit individual(s), as selecting only the most elite individuals can limit exploration (Eiben and Smith 2015). A more complex example of elitism is found in the popular GA NSGA-II, where the individuals are organized into (Pareto) non-dominated fronts and are selected iteratively from these fronts, ensuring that the fittest individuals remain (Deb et al. 2002).

Termination

Most EAs simply terminate after a pre-defined number of generations (or fitness evaluations) have passed, though performance-based criteria can also be utilized. For example, no significant change in fitness for a certain number of generations or evaluations can indicate convergence and terminate the EA (Jain, Pohlheim, and Wegener 2001), though there are many ways (discussed in Section 3.2.3) to measure both the change itself and its significance (Trautmann et al. 2008; Trautmann et al. 2009). Alternatively, thresholds can be incorporated such that the EA terminates when a pre-defined objective function value has been reached. While this component is generally given less consideration than the others in algorithm design, the choice can strongly influence the quality of the final population (Ghoreishi, Clausen, and Jørgensen 2017).

3.2.3 Performance analysis

After termination, there are two main challenges facing practitioners: *performance analysis* and *solution selection*. These are issues that primarily exist for multi-objective optimization, where it can be unclear which individuals are best. Analyzing the performance of an EA may be through the perspective of convergence (such as the fluctuation of fitness across generations, or number of evaluations required for convergence) or the quality of the final population (thus enabling performance comparison between different algorithms or parameters). In both perspectives, measures are needed that quantify performance in some way in order to be used for comparing different algorithms, runs, parameters etc. For further details, see Zitzler et al. (2003) and Zitzler, Knowles, and Thiele (2008).

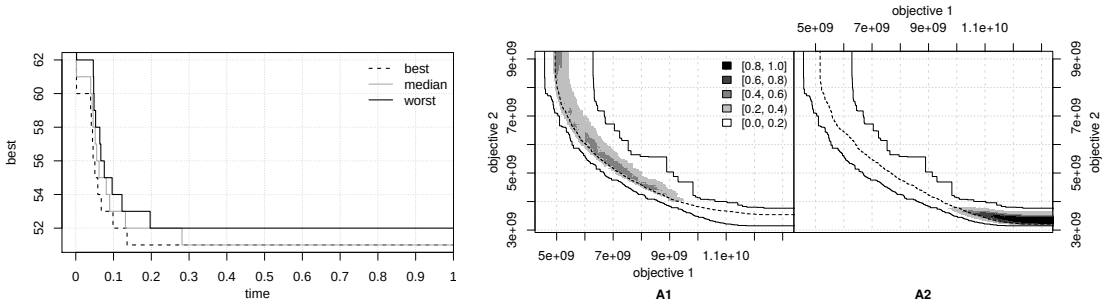
In reality, the final population produced from an EA is an *approximation* of the Pareto front, as it can be impracticable or even infeasible to generate the complete front (Zitzler et al. 2003). Due to this, and the inherent stochasticity of EAs, different runs can produce different final populations (also referred to as *approximation sets*). Thus, it can be useful to not only compare different algorithms or different parameter configurations for the same algorithm, but also the variability or robustness of a single algorithm configuration over multiple runs.

Owing to its extensive use in the community and later in this thesis (Section 7.2.5), the hypervolume is one example of a *unary* measure of performance (Zitzler and Thiele 1998). It is a measure of the volume of the objective space dominated by a given set of solutions, thereby providing a measure of both the diversity of the population and the extent of optimization for each objective. By using a shared reference point (also known as the nadir point) to calculate the volume, multiple configurations or algorithms can be directly compared. This measure is Pareto-compliant, providing a strictly better value for an approximation set when it dominates another (Zitzler et al. 2003).

Beyond comparing singular values, empirical attainment functions (EAFs) have been proposed to probabilistically analyze and compare approximation sets, permitting visualization of the (estimated) probability that areas of the objective space will be dominated by solutions producing during a run (López-Ibáñez, Paquete, and Stützle 2010). Due to the stochastic nature of (in this context) EAs, each approximation set will vary between runs. The attainment function aims to define the distribution from which these random sets originate, and is defined as the probability of a set of non-dominated solutions, $\mathcal{X}'$, *attaining* (i.e. generating a solution that weakly Pareto-dominates) an arbitrary point $\mathbf{x}' \in \mathbb{R}^n$. Attainment of this point is denoted $\mathcal{X}' \trianglelefteq \mathbf{x}'$ (Fonseca et al. 2011). Mathematically, the attainment function is the probability that $\exists \mathbf{x} \in \mathcal{X}' : \mathbf{x} \succeq \mathbf{x}'$. For j non-dominated approximation sets $\mathcal{X}'_1, \ldots, \mathcal{X}'_j$, the EAF is the function $\alpha_j : \mathbb{R}^n \to [0, 1]$ that is defined as:

$$\alpha_j(\mathcal{X}'_1, \ldots, \mathcal{X}'_j; \mathbf{x}') = \frac{1}{j} \sum_{i=1}^{j} \mathbb{1}(\mathcal{X}'_i \trianglelefteq \mathbf{x}') \tag{3.4}$$

where $\mathbb{1}(\cdot)$ is the indicator function that is 1 if the condition is true and 0 otherwise. Note that this is the *empirical* attainment function as the limited number of approximation sets provide an estimate of the attainment function (Fonseca



(a) EAF plot (b) EAF difference plot

Figure 3.6: Example of an empirical attainment function (EAF) plot and EAF difference plot generated from the *EAF R package*.

et al. 2011).

EAFs can then be used for visualization through the generation of "k%-attainment surfaces" (Fonseca and Fleming 1996; López-Ibáñez, Paquete, and Stützle 2010), which represent a separation in the objective space highlighting the area where solutions would have been attained in at least k% of the runs. These surfaces can then highlight expected e.g. worst-case or median performance (exemplified in Figure 3.6a). Further information on the computation of the EAF and the attainment surfaces can be found in Fonseca et al. (2011).

An EAF difference plot (López-Ibáñez, Paquete, and Stützle 2010) can directly compare the search capabilities of two methods in the objective space. As can be seen in Figure 3.6b, the shading in different parts of the space represents the probability of that approach to find solutions in that region compared to the other approach. This example shows that the second method ("A2", on the right) is far superior at minimizing the second objective only, but the method "A1" has slightly better performance across a larger part of the front for both objectives.

The selection of a single solution from the set is generally problem-specific, and is an open area of research. For a practical application a domain expert can ideally select a solution according to their own personal criteria and expertise. Preferences can be used to weight the objectives, enabling easier selection.

3.3 Constraint handling techniques

For many real optimization problems, we have constraints that restrict the decision space between feasible and infeasible regions (as highlighted in Figure 3.1), limiting the range of values that can be explored for each decision variable. This

section discusses some constraint handling methods for use in GAs. For further details see Coello Coello (2002), Mezura-Montes and Coello Coello (2011), and Michalewicz and Schoenauer (1996).

Constraints can exist as real limitations for a system. For example, in the optimization of the placement of buoys for power generation (Neshat et al. 2019), the constraints are inherent limitations for the actual equipment being optimized. In Chugh et al. (2019b), optimizing the design of a ventilation system is constrained by manufacturing limitations. Such constraints, known as box or domain constraints, limit the values of individual decision variables.

The inequality (g_i) and equality (h_j) constraints (Equation 3.1) can themselves be linear or non-linear, further restricting the possible values for decision variables and affecting the complexity of the search. Though traditionally these constraints are imposed by an externality (design constraints, practicalities etc.), in Section 5.1 we use constraints more akin to secondary objectives in order to embed a preference towards the solutions with degrees of constraint violation (which represent an aspect of problem difficulty).

The need for specific constraint handling techniques originates from the fact that, in their base form, EAs do not explicitly consider constraints (Michalewicz and Schoenauer 1996). That is, neither fitness functions nor genetic operators are guaranteed to create feasible individuals and satisfy constraints. Thus, specific techniques are needed to embed consideration of the constraints and feasibility into the algorithm without disrupting the search. As the optimal solution may lie close to the boundary of the feasible region, exploration of the infeasible region may be necessary for discovering (near-)optimal solutions.

Two popular taxonomies have been proposed for constraint handling techniques: Michalewicz and Schoenauer (1996) and Coello Coello (2002). Mezura-Montes and Coello Coello (2011) tried to unite these taxonomies to better frame more recent research, identifying broad categories into which the techniques fall: penalty functions, decoders, special operators, and the separation of objective and penalty functions. Specific attention will be given to the first and last of these categories as they are most relevant to this thesis.

3.3.1 Penalty functions

Penalty functions are used to worsen the fitness, so as to preferentially select (more) feasible solutions. The general formula for a penalty function is:

$$\phi(\mathbf{x}) = f(\mathbf{x}) + p(\mathbf{x}), \tag{3.5}$$

where $\phi(\mathbf{x})$ is the penalized objective function and $p(\mathbf{x})$ is the penalty function. The quadratic loss function is one example of such a penalty function (Runarsson and Yao 2000), which is calculated as follows:

$$p(\mathbf{x}) = \sum_{i=1}^{m} r_i \cdot \max\left(0, g_i(\mathbf{x})\right)^2 + \sum_{j=1}^{p} c_j \cdot |h_j(\mathbf{x})|, \tag{3.6}$$

where r_i and c_j are penalty factors for each constraint. It should be noted here that $p(\mathbf{x})$ is positive for our aforementioned minimization problem, but the sign can be flipped (for maximization) with no loss of generality of Equation 3.5.

Penalty factors directly influence the magnitude of the penalty being applied, and therefore static values are specific to the magnitude of the fitness values for the problem. As a result, such fixed values may be unhelpful or even misleading during optimization (Runarsson and Yao 2000; Mezura-Montes and Coello Coello 2011). The magnitude of these penalty factors can control whether a constraint is considered "hard" or "soft". Hard constraints must not be violated, and can therefore use a factor large enough to ensure this is respected (known as a "death penalty"). Such immediate removal of infeasible solutions can lead to poor exploration, however, and is thus unfavourable (Yeniay 2005). Lower penalty factors can be used for soft constraints, enabling their use as preferences that do not need to be satisfied, but an ideal solution would violate these less (or not at all if possible).

The use of dynamic penalty functions can help use more appropriate penalty values at different times of the optimization. Of course, the use of such functions introduces additional parameterization which can also be problem-dependent. A simple example of such a function is to incorporate the generation counter into $p(\mathbf{x})$, either as a linear multiplier (Joines and Houck 1994) or by using more complex functions (Kazarlis and Petridis 1998). Similarly, which function is appropriate to use may also vary, making it difficult to find generally applicable methods. Regardless, such dynamic techniques are generally predicated on the

idea that lower penalization at the beginning assists with exploration, and feasible solutions become increasingly preferred over time, effectively utilizing the explore-exploit idea.

To incorporate information about how the search is currently performing (and thus using more problem-dependent information), adaptive penalty functions have been proposed. These can use the fitness of the best solution found in a previous number of generations (Rasheed 1998), or simply the current ratio of feasible to infeasible solutions to balance the search (Hamida and Schoenauer 2002). Mezura-Montes and Coello Coello (2011) note that the downside of adaptive methods is the potential to make changes (to the search) that may be unhelpful in later stages of the optimization.

For the comparison of individuals based on their constraint violation, fair comparison across constraints can become an issue. Constraints of different magnitude both to each other and to the fitness (when combined as in Equation 3.5) can result in a bias towards the numerically larger constraint. As Paula Garcia et al. (2017) note, methods such as normalization are not always possible for some applications (e.g. where the bounds are unknown). The magnitude of these constraints can even distort the fitness landscape itself, particularly for static penalties (Equation 3.5) that combine the penalty and objective functions (Deb 2012).

3.3.2 Separation of objectives and constraints

By separating the objectives and constraints, the problem can be transformed to allow the use of multi-objective optimization methods for handling trade-offs, creating a more balanced search between them. The two main ways to transform the problem are: formulating a bi-objective problem (typically using the original fitness function and sum of constraint violation); and, turning the constraints into objectives (Mezura-Montes and Coello Coello 2011). The latter has the issue that multiple constraints result in a *many*-objective problem, where traditional comparison methods, such as Pareto dominance, become less useful as a greater percentage of the objective space becomes non-dominated (Ishibuchi, Tsukamoto, and Nojima 2008).

One popular technique in this category was originally proposed in Deb (2000) and sets three feasibility criteria/rules to a binary tournament selection: two

feasible solutions are compared by their fitness; a feasible solution is always preferred over an infeasible one; and, two infeasible solutions are compared by their sum of constraint violation(s). The concept of using different methods to compare feasible and infeasible solutions has also been used by stochastic ranking (Runarsson and Yao 2000) and other related ranking methods (Ho and Shimizu 2007; Paula Garcia et al. 2017; Runarsson and Yao 2002).

Stochastic ranking

Owing to its extensive use in HAWKS (Chapters 5 and 6), further detail is given on the stochastic ranking method. Introduced by Runarsson and Yao (2000), stochastic ranking attempts to balance the search between the fitness and penalty functions by probabilistically using either one when comparing individuals. The aim of the method is to rank (and sort) the individuals for either parental or environmental selection.

Adjacent individuals in the population are iteratively compared. Two feasible individuals are always compared using their fitness. Otherwise, the two individuals are compared using either the fitness or constraint violations with a probability of P_f and $1 - P_f$ respectively. The winner, judged by either a better fitness or a lower constraint violation, is then placed at a higher rank in the population (i.e. the winner is swapped if need be), using μ sweeps through the population (or stopped early when no swaps have been made during a sweep)[1]. Pseudocode for the algorithm can be found in Algorithm 3.1, where a population of μ individuals is sorted. If used for environmental selection, the combined pool of $\mu + \lambda$ individuals would instead be sorted. For further details or alternative pseudocode, see Runarsson and Yao (2000).

From their experiments, Runarsson and Yao (2000) recommend $0.4 \leq P_f \leq 0.5$, though B. Li et al. (2016) found $0.4 \leq P_f \leq 0.6$ to be more useful. When all solutions are infeasible, the selection becomes a weighted bi-objective problem of the fitness and sum of constraint violations with weights of P_f and $1 - P_f$ respectively. Ultimately, the appropriate value of P_f is both problem- and algorithm-specific, as it represents a bias in the search that may be more appropriate depending on the nature of the fitness and penalty functions or the fitness landscape of the problem (where, for example, the optimum lies on the edge of the feasible region). In Chapter 5, we use this parameter to directly encode a preference into

[1] This is essentially a bubble-sort-like procedure (Runarsson and Yao 2000).

Algorithm 3.1: Stochastic ranking

input : P_f, μ, $\mathbf{x}_i \; \forall \; i \in \{1, \ldots, \mu\}$
output: Individuals sorted by rank

```
for i = 1 to μ do
    for j = 1 to μ − 1 do
        u ∼ U(0, 1)
        if (p(x_i) = p(x_{i+1}) = 0) or u < P_f then
            if f(x_i) > f(x_{i+1}) then
                Swap(x_i, x_{i+1})
        else if p(x_i) > p(x_{i+1}) then
            Swap(x_i, x_{i+1})
    end
    if no Swap then
        Break
end
```

the search where infeasible solutions are desirable, making all possible values of P_f potentially useful in contrast with its traditional use.

3.4 Parameter setting & self-adaptation

An issue of EAs is that the various components (operators, selection mechanisms etc.) typically have hyperparameters that affect the ability of the algorithm to find (near-)optimal solutions. Compounding this, the complex, epistatic interactions between these many parameters makes the use of more simplistic methods (e.g. factorial design) of limited use when determining ideal parameters, which themselves will vary by problem for the same algorithm.

In isolation, different components of the algorithms have their own suggested parameter settings (such as length-based mutation probabilities), or off-the-shelf operators, yet combined these may not be suitable, and/or they may not be suitable for the particular problem. While the field of hyperparameter optimization has seen increasing prominence in the machine learning community (Feurer and Hutter 2019), the use of these methods (from grid search to Bayesian optimization) in the evolutionary computation community is less widespread (Karafotias, Hoogendoorn, and Eiben 2015).

In this section, we discuss parameter tuning and parameter control, and some methods that are used to adapt the parameters of an EA during optimization to avoid setting values *a priori*. This will provide background necessary for the work in Chapter 7. For a more complete review, see Eiben, Hinterding, and Michalewicz (1999), Eiben and Smit (2011), and Karafotias, Hoogendoorn, and Eiben (2015).

3.4.1 Parameter tuning & parameter control

The *parameter tuning problem* is the need to select good values for the hyperparameters of the EA itself (such as the population size), whereas the *parameter control problem* is the fact that for ideal performance these values may need to be varied *during* optimization. Ignoring implementation practicalities, solving the control problem implicitly solves the tuning problem, though these can be designed and used independently (Karafotias, Hoogendoorn, and Eiben 2015). The strategies for tackling these two problems are multifarious, and the relative merits of *tuning* versus *control* are both algorithm- and problem-dependent, and as such lack generalized approaches.

In Chapter 7, we consider a parameter control problem where the search space itself is adapted by progressively expanding the genotype based on whether the hypervolume indicates convergence. For an overview of parameter tuning methods, see Eiben and Smit (2011). As the selection of good hyperparameters for an algorithm to a specific problem relates to hyperparameter optimization, this is further discussed in Section 4.2.1 with the additional context of clustering, as the methods there are not specific to EAs but to the wider problem of selecting appropriate parameters.

Parameter control

To better place our work in Chapter 7 into context, we provide some further background on parameter control. Eiben, Hinterding, and Michalewicz (1999) tried to unite previous classification schemes, resulting in the main criteria for method classification being based on *what* component of the EA was changed, and *how* the change was made. Similar to Hinterding, Michalewicz, and Eiben (1997), parameter control methods were subdivided into *deterministic*, *adaptive*,

and *self-adaptive* categories. *Deterministic* methods make a change to a parameter using some pre-defined rule e.g. a change occurs every set number of generations. *Adaptive* parameter control methods utilize some feedback from the search to alter the change (magnitude and/or direction) of a parameter. *Self-adaptive* methods incorporate the parameters themselves into the encoding, and therefore directly undergo evolution in the hope that good values result in fitter individuals and thus propagate. More recently, Karafotias, Hoogendoorn, and Eiben (2015) provided a more focused review on parameter control, as this is the harder problem that has seen less progress than parameter tuning.

Methods from all three categories have been applied to different components of EAs, of which we discuss some *adaptive* methods here (as it is most relevant to our work). For the representation, the ARGOT (adaptive representation genetic optimizer technique) strategy proposed in Shaefer (1987) adapted the search space by changing the number of bits (i.e. the numerical precision) in the genotype. Schraudolph and Belew (1992) adapted the genotype-phenotype mapping directly to change the resolution of the search, effectively narrowing the search space. The trigger that changes the resolution is based on a count of the number of individuals in different numerical ranges, after enough individuals are in this range the search is considered to have converged and thus the resolution can narrow further. Some of these strategies are further discussed in relation to our work in Section 7.2.1.

Setting mutation probabilities has received much attention due to its strong impact on performance, yet typically heuristics (calculating based on the length of the genotype) or constant values are used (Eiben, Hinterding, and Michalewicz 1999; Grefenstette 1986; Mühlenbein 1992). Other works do not use a static mutation rate, however, by using e.g. a deterministic control scheme to decrease the rate over time (Fogarty 1989) or an adaptive control scheme based on performance (Cobb 1990) can help either explore or fine-tune as appropriate. This latter work monitored the average fitness of the population and, after a drop in this average, employed *hypermutation* (where the mutation probabilities are significantly increased for a single generation). This short-term rapid exploration is intended to help find new, useful genetic material, which is particularly useful if the search space itself changes (and thus there is an abundance of new genetic material to find). As a result, we adopt this method in Section 7.2.1.

While Cobb (1990) used the average fitness to determine whether any change

was required, Martínez, Oropeza, and Coello (2011) use the hypervolume to detect convergence and thus trigger a change in the population size. The authors do not discuss precisely how they determine whether there is no improvement in the hypervolume, so it is assumed that a direct comparison of the exact hypervolume is used (as opposed to e.g. a threshold). In Section 7.2.2 we measure the rate of change of the hypervolume instead, so that a change is triggered before the current search space has been completely explored.

One framework, "Borg", embeds convergence detection and strategies to deal with it into the GA (Hadka and Reed 2013). In contrast to our aforementioned work, which uses the rate of change in the hypervolume, Borg divides the objective space into ϵ-boxes (Deb, Mohan, and Mishra 2005), where progress is measured as the time since a solution has been found in a better (i.e. Pareto-dominant) box. If progress is not found, several restart mechanisms are employed (adapting the population size, tournament size, and re-population from the elitist archive). This framework also uses multiple crossover and mutation operators which are adaptively selected based on which produce a higher proportion of dominant individuals. Such self-adaptive approaches add significant complexity, but may confer flexibility in selecting appropriate hyperparameters for each individual problem.

The efficacy of controlling the parameters is related to the purpose of the EA. For example, adapting the mutation rate may help lead to a more diverse search if insufficient diversity is expected, but this has limited use if it does not assist the desired outcome (such as focused search on a region of the Pareto front). Such considerations can help when deciding what parameters require tuning and how to tune them.

3.5 Summary

This chapter has introduced single- and multi-objective optimization problems (Section 3.1), and a class of algorithms that can be used to solve these problems (Section 3.2). We have introduced various important areas of research within EAs pertinent to our work, namely constraint-handling techniques (Section 3.3) and methods for parameter control and tuning (Section 3.4). In the next chapter, we introduce challenges and methods for producing synthetic data; later in Chapter 5 we use these EAs to generate such data.

Chapter 4

Background: Evolving Synthetic Data

This chapter provides background information relevant to the synthetic data generator we propose in Chapters 5 and 6. We first discuss the need for synthetic data and its role in experimentation in Section 4.1. We then discuss the algorithm selection problem in Section 4.2 and how synthetic data is useful to tackle it, followed by a review of different synthetic data generators in Section 4.3. Some of the content in this chapter has been adapted from Shand et al. (2019).

4.1 The need for synthetic data

With regards to terminology, we distinguish between synthetic and real-world data by the former having a constructed/defined generating mechanism (e.g. sampling from a defined distribution), and the latter being collected (e.g. from sensors or observations), representing samples from one or more typically unknown distributions/models. Of course, the collection and processing of real-world data can clean the data such that it can practically function as synthetic data, but for our purposes we distinguish the two by their source.

Empirical comparison between techniques is a cornerstone of the scientific method. At a community level, methods developed by independent researchers need to be compared in order to gain insights into the applicability and efficacy of their developments. As newly proposed methods must be compared to existing ones on the same data, subsequent research is highly likely to use these same datasets. Difficulties with reproducing previous work further enforces the use

of these same datasets to facilitate comparisons. This feedback loop results in "standard" datasets becoming virtually required to include in experimentation (Hooker 1995). This reliance on experimenting with a subset of datasets also occurs more explicitly through the creation of benchmark suites — a collection of problems collated and/or created for the purpose of widespread comparison. This issue can occur with either synthetic or real-world data.

The issue of this feedback loop is that the community as a whole tunes both hyperparameters and algorithmic development to these specific problems (Hooker 1995; Schmidhuber 2015). If these popular problems represent a broad spectrum of challenges that reflect real-world challenges, then this is not a negative — analyzing whether these problems adequately cover the space of encounterable problems is difficult, if even possible to do in its entirety (Recht et al. 2019). Hooker (1995) argues for "controlled experimentation" i.e. comparing algorithmic performance on a specific problem characteristic that the research in question is addressing, compared to the "competitive testing" that is encouraged when the same subset of datasets are re-used time and again, often with a limited pool of comparison measures.

Comparing performance using data that explicitly exhibits the problem characteristic under investigation is inherently simpler with synthetic data, as we have control over the generating mechanism and thus (to varying extents) the properties of this data. The use of real-world data for this purpose is possible if there is an appropriate measure of the problem characteristic and a controlled way to vary this. For a concrete example, if we are measuring the ability of algorithms to scale with the size (e.g. dimensions) of the data, then this is simple regardless of the data source as we can simply sample data to the desired size. If, however, we are investigating the robustness of clustering algorithms to clusters in close proximity then both a measure of the proximity and control over this proximity is vital to identify differences, which requires a purposefully constructed generating mechanism.

Insights into the algorithms beyond binary superiority (i.e. algorithm A is better than algorithm B) are obtained under the assumption of the No-Free-Lunch (NFL) theorem, which supports the intuition that no single algorithm is expected to be superior across all problems (Wolpert and Macready 1997). McDermott (2020) reviews the arguments for and against the validity of the NFL theorem (and its subsequent refinements) in different contexts, highlighting how

the NFL is often mis-used. In clustering it is demonstrably clear, however, that no single algorithm is superior for every problem (as outlined in Chapter 2 and discussed in Ben-David [2018]). The inductive bias of each algorithm (e.g. the assumption of GMM that the data is Gaussian) differs, but fundamentally limits their capabilities. Thus, having a diverse range of datasets is necessary to reveal particular algorithms' strengths and weaknesses, and that the diversity of these datasets is understood and (ideally) explicitly controlled. Hooker (1995) argues that benchmark problems should be constructed to control for parameters that may affect performance, rather than problems that purely represent reality (thus potentially conflating insights). This supports the utility of toy datasets, of which many have been created for clustering (Fränti and Sieranoja 2018; Handl and Knowles 2006) due to their simplicity and easy visualization of results. Although these datasets may serve to illustrate simple capabilities or properties of clustering algorithms (such as a broad favouring towards compactness or connectedness, as discussed in Section 2.3), these scenarios are often too contrived to consider more than a single, basic characteristic and thus provide more complex challenges.

When assessing the diversity of benchmark problems, we refer to the specific properties/challenges of the datasets, such as those discussed in Section 2.3. Macià and Bernadó-Mansilla (2014) analyzed the datasets in the popular UCI Machine Learning Repository (Dheeru and Karra Taniskidou 2017), finding a surprising similarity in complexity (in terms of both statistical measures of the data and the limited diversity of classification performance for algorithms with the same parameters) across them. We expect that a diverse set of problems would require not only a tuning of algorithm-specific hyperparameters, but a vast range of performance in the algorithms (that for a group of sufficiently competent algorithms should not positively correlate). Without explicit effort to identify and quantify aspects of problem structure or difficulty, it is difficult to ensure a diverse set of problems and thus a comprehensive benchmark suite. Such a task is further complicated in clustering, due to the subjective nature of the task itself (Section 2.1).

The utility of including real-world problems in a suite is undeniable due to the inherent complexity of real-world data and, fundamentally, the fact that it is real-world data to which algorithms will be applied. The performance on this data likely reflects their *actual* applicability and efficacy. As discussed in Section 2.1,

Luxburg, Williamson, and Guyon (2012) argue that the utility of clustering lies only in its domain-specific application, further supporting the inclusion of such problems. Note that this data has to be suitable for clustering, however, as it is an assumption that class labels are synonymous for cluster labels (Guyon, Von Luxburg, and Williamson 2009). Meaningful cluster labels facilitate external cluster validation (Section 2.5), which is required for an unbiased evaluation of performance. Given this, synthetic benchmarks are often preferred over the use of real-word data as they offer an opportunity to accurately evaluate performance on them, in addition to the ability to explicitly model problem properties and exercise control over these properties (Smith-Miles and Bowly 2015).

4.2 The algorithm selection problem (ASP)

The algorithm selection problem (ASP) was first introduced by Rice (1976), which seeks to predict which algorithm (from a portfolio) will perform best on a problem given a set of measurable problem features. There is inherent motivation in the concept of identifying the right algorithm for the problem at hand, which Smith-Miles (2008) argues is further motivated by the NFL (Wolpert and Macready 1997). The bias of clustering algorithms to the discovery of a subset of possible cluster structures (Section 2.4) naturally lends itself to a formulation of the ASP. In this section, we define the ASP and the challenges involved with its application.

We first define notation for the ASP, as used in Smith-Miles and Lopes (2012):

- The problem space ($\mathcal{P}$) is the set of instances of a problem (for our purposes, an instance is synonymous with a dataset).
- The feature space ($\mathcal{F}$) contains the set of measurable properties of the instances (referred to as *problem features*).
- The algorithm space ($\mathcal{A}$) is the portfolio of algorithms used to solve the problem.
- The set of performance measures ($y \in \mathcal{Y}$) that map the result of an algorithm for a given instance to a numerical value.[1]

[1]Note that Smith-Miles and Lopes (2012) explicitly define the performance space as $\mathcal{Y}$, whereas in Rice (1976) the performance measures simply map to $\mathbb{R}$.

Thus, the ASP can then be defined as: for a given problem instance $\mathbf{x} \in \mathcal{P}$, with a feature vector $f(\mathbf{x}) \in \mathcal{F}$, find the selection mapping $\mathcal{S}(f(\mathbf{x}))$ into $\mathcal{A}$, such that algorithm $\alpha \in \mathcal{A}$ maximizes the performance measure(s) $y(\alpha, \mathbf{x}) \in \mathcal{Y}$.

The utility of any prediction of algorithmic superiority on problem instances is predicated on the descriptive power of the feature set ($\mathcal{F}$). The identification of appropriate or relevant features that form such a feature set is both difficult and domain-specific. The difficulty of selecting the appropriate feature set has naturally led to work such as Muñoz et al. (2018), where they used feature selection[2] to identify a relevant subset of features from 509 different measures (from simple statistical values, to information-theoretic measures, to itemset rules) of the data. The use of feature selection has its own downsides, such as the bias introduced when identifying the relevant subset, or the need to re-run feature selection should the collection of features change (which may be desirable for iterative improvement as new features are added). A more complex approach was used in Smith-Miles and Bowly (2015), where a genetic algorithm was used alongside a classifier to identify the best subset of features to use with PCA (discussed in Section 2.2.1). Regardless of the approach used, when there are many possible features that could be measured, such an approach may be needed to identify both useful and (ideally) uncorrelated problem features to construct the instance space (discussed in Section 4.3.1).

4.2.1 The ASP in other contexts

The desire to select the best algorithm for a given task exists in other areas, though not necessarily using the same formulation as Rice (1976). Smith-Miles (2008) attempted to unite multiple sets of disparate literature that, at their core, relate to the ASP. From *meta-learning* in the machine learning community to *landscape analysis* in the metaheuristics community, the ability to predict performance of algorithms on a given problem is a focus of research. Smith-Miles (2008) argues that this disconnect has slowed research in this area, which is merited, yet as we previously discussed a core component of this research is the identification of problem features ($\mathcal{F}$) which is both complex and domain-specific and as such limits broader progress.

[2]Feature selection is the process of selecting a subset of data features with the aim of reducing the dimensionality, removing irrelevant features that do not help with classification/pattern recognition, and/or removing redundant features which do not provide additional information over existing features.

Meta-learning, or *learning to learn*, is "the science of systematically observing how different machine learning approaches perform... and then learning from this experience" (Vanschoren 2019). With reference to algorithm selection, it is learning the selection mapping $\mathcal{S}(f(\mathbf{x}))$ into $\mathcal{A}$. The current meta-learning paradigm looks at the prediction of *task* (synonymous with instance) performance for a given configuration (e.g. hyperparameters). The shift of approach in predicting configurations (as opposed to a particular algorithm from a portfolio) is relevant for the task of hyperparameter optimization, which is vital in machine learning. Note that this approach is synonymous with the "algorithm configuration problem", which has been considered in the optimization community (Birattari 2009; Hutter, Hoos, and Leyton-Brown 2011; López-Ibáñez et al. 2016). The selection of appropriate hyperparameters is also vital for metaheuristics (referred to as the *parameter tuning problem* which we discussed in Section 3.4.1), for which approaches such as that detailed by López-Ibáñez et al. (2016) is focussed on (though their framework, `irace`, is agnostic in application).

In this thesis we are more concerned with the identification of problem features that describe datasets, rather than the algorithm selection itself. In the meta-learning field, "meta-features" are used analogously as problem features, which can range from simple statistics (Michie, Spiegelhalter, and Taylor 1994) to identifying task similarity directly (Bardenet et al. 2013). We discuss problem features further in Section 4.2.2. Overall, further developments in this field provide potential opportunities to the broader work on algorithm selection, thus making the link between them important.

Similarly, landscape analysis (or ELA, exploratory landscape analysis) is an area trying to quantify different features of the fitness landscape in order to both identify the "best" algorithm, but also to improve understanding of both algorithms and their relationship to different problems (Mersmann et al. 2011). It is an area that is growing in its own right, as evidenced by its introduction as a tutorial at GECCO (the *Genetic and Evolutionary Computation Conference*) in 2017 (Kerschke and Preuss 2017) that has continued since. Recent work (Kerschke and Preuss 2017; Kotthoff et al. 2015) directly references the algorithm selection problem, highlighting that the previous issue of divergence literature identified in Smith-Miles (2008) may no longer be as prevalent an issue. Software such as `flacco`[3] help both in the compilation of problem features, and in their application

[3] http://kerschke.github.io/flacco/

to a wider set of fitness landscapes (thus providing further insights). The recent creation of the "Benchmarking Network"[4] holds promise for further development of this field, following consideration of Hooker's discussion around "competitive testing" that benchmarking can encourage (Hooker 1995).

Our work in Chapters 5 and 6 is motivated by the need for identifying a set of problem features for clustering and a generator that can create instances with varying properties, which is required for the ASP to be investigated for clustering. For this, we build upon the body of work from Smith-Miles, which we provide further background on in Section 4.3. As discussed in the future work (Section 8.2.1), the developments in these other fields may be useful for further extending our work in clustering.

4.2.2 Problem features

The ability to accurately predict which algorithm from the portfolio ($\mathcal{A}$) is best suited to a particular problem is entirely predicated on the descriptive power of the problem feature set ($\mathcal{F}$), specifically in relation to the capabilities of the algorithms. For example, a set of features that describe (in different ways) the elongation of the clusters in the dataset is meaningless if the algorithms are mostly agnostic to variations of this (such as single-linkage). Thus, not only is the identification of problem features dependent upon the domain or type of problem (i.e. problem class), but also potentially even on the algorithms available.

As previously mentioned, the field of meta-learning discusses "meta-features" which are synonymous with the problem features we consider in this thesis. A more complete overview of meta-features is given in Castiello, Castellano, and Fanelli (2005) and Vanschoren (2019). In general, they fall under the similar categories of work discussed later in this section; these meta-features are primarily statistical or information-theoretic measures of the data, which are not found to be uniformly descriptive across datasets for predicting performance of classification algorithms (Bilalli, Abelló, and Aluja-Banet 2017).

The need for the problem features to be application-/domain-specific is highlighted in the work of Smith-Miles (and co-authors). They have tackled the ASP on problems such as combinatorial optimization (Smith-Miles and Bowly 2015), classification (Muñoz et al. 2018), and outlier detection (Kandanaarachchi, Muñoz, and Smith-Miles 2019). For each, a different set of problem features was

[4] https://sites.google.com/view/benchmarking-network/home

identified, with very little overlap between the sets of features (only standard statistical measures, such as correlation or skewness of the data, were common). As such, in addition to its importance, it is clear that the selection of problem features is a non-trivial task.

The esoteric nature of the differences between typical measures of clusters (such as the cluster validity indices discussed in Section 2.5) further exacerbate this difficulty for our problem domain. As previously noted, the study by Arbelaitz et al. (2013) indicated that while some of these indices do share similarities, they do not consistently correlate between different types of clustering problem. Ferrari and Castro (2015) looked at the algorithm selection problem for clustering within the context of meta-learning, selecting "meta-attributes" (another term for problem features) under three categories: *simple*, such as the size of the dataset; *statistical*, such as the average correlation between features; and, *informational*, such as the entropy of the discrete features. Soares, Ludermir, and A. T. de Carvalho (2009) used a similar set of features to predict a ranking of clustering algorithms on synthetic data (generated using the method proposed in Handl and Knowles [2005b], which we discuss later in Section 4.3.2).

In both of these works, no problem features were used that were specific to clustering (or related to the properties discussed in Section 2.3, such as compactness or connectedness), and were evaluated using datasets from the UCI repository (Dheeru and Karra Taniskidou 2017) which, as previously discussed, may not be suitable for clustering (Luxburg, Williamson, and Guyon 2012). Despite the limited scope of the experiments, there were promising results that a ranking of clustering algorithm performance could be predicted. Our work experiences a similar difficulty in identifying suitable problem features (Section 5.2.4), though in Section 6.2.1 a wider set of features is used that takes into account a more varied perspective of different cluster properties.

4.3 Instance generation

In this section, we focus on research that has investigated generating problem instances. In particular, we discuss methods for visualizing the instances, generating instances, and finally a discussion about generators used specifically for clustering and the general properties that such a generator should have.

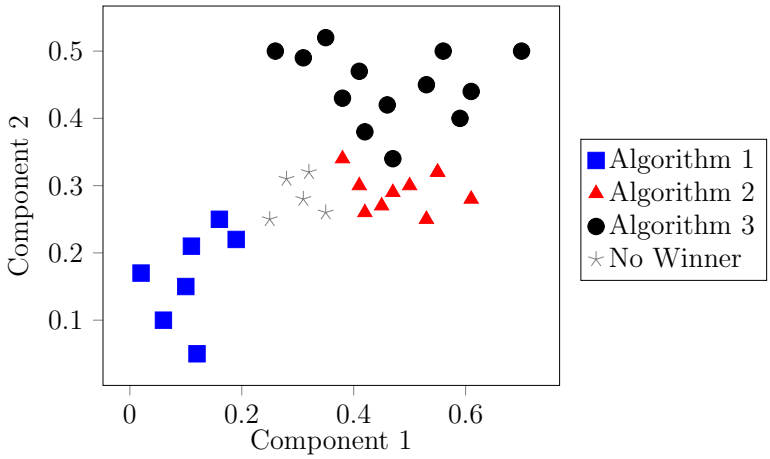


Figure 4.1: Illustration of an instance space, with the winning algorithm highlighted to show the 'footprints' (areas where a particular algorithm is dominant).

4.3.1 Instance space

There have been multiple works that look to extend Rice's framework for the ASP (Oliveira et al. 2018; Smith-Miles et al. 2014; Smith-Miles and Bowly 2015; Muñoz et al. 2018). A key part of this work is the creation of an *instance space* (Smith-Miles and Tan 2012), which is a visualization of the identified problem features, thus allowing the visualization of the instances. This facilitates observation of the problem diversity (according to the set problem features, $\mathcal{F}$), highlighting where there may be gaps i.e. datasets with properties that do not yet exist. Having a visualized space of the datasets also allows for the identification of an algorithm's 'footprint' (an area where a particular algorithm is dominant), potentially providing insights into its strengths and weaknesses (Corne and Reynolds 2010). In order to create a comprehensive benchmark suite, an understanding of the diversity among the existing datasets is paramount to robust evaluation of algorithms.

An example of an instance space is shown in Figure 4.1, showing a number of datasets plotted, which are highlighted by the algorithm that performed the best on them. This is an ideal instance space, where there is a clear footprint for each algorithm, i.e. a region in the space for which a particular algorithm is uniquely suited. In practice, to obtain an instance space such as this, the problem features and algorithms must be quite distinct (which is unlikely for both complex tasks and algorithms). As shown in this example, there may be datasets for which there is no winner (which may be due to their simplicity, equivalent difficulty to

different algorithms, or a consequence of the scoring method being used).

Typically, PCA is used to project the problem feature set ($\mathcal{F}$) down to the visualizable two dimensions. Whether PCA is the appropriate method for this depends on the datasets, number of features, and how much variance is lost in the projection, though in the literature it has been applied to tasks with three problem features (Smith-Miles and Bowly 2015) up to 40 (Smith-Miles and Tan 2012). Although useful, as PCA is inappropriate for non-linear relationships, for complex problem features sets it may not be the best approach. In Muñoz et al. (2018), the authors devised a projection method (later referred to in Kandanaarachchi, Muñoz, and Smith-Miles [2019] as 'Prediction Based Linear Dimensionality Reduction', or PBLDR) that explicitly incorporates algorithmic performance into the projection, such that the resulting space is an optimal projection of the datasets according to their difficulty and problem features rather than just their problem features alone. Although useful, the complexity of this method (involving different transformations for the features, running a random forest model for each algorithm on each possible feature set combination, and obtaining the optimal projection using BI-population CMA-ES [Hansen 2009]) adds a barrier for usage. The use of more complex projection methods can also remove the intuition that PCA provides (i.e. the correlation between features) which can help when identifying problem feature values that are not currently exhibited.

Note that, although 'instance space' refers to a specific construction used in the aforementioned works, other related works have also used visualization techniques to understand the diversity of data using slightly different methods. In Macià, Orriols-Puig, and Bernadó-Mansilla (2010), they visualize both real-world and derived synthetic problems onto a "complexity space" using different measures of problem complexity. All plots used in this work are pairwise, so despite having multiple such measures only two are ever simultaneously visualized. While this provides more fine-grained information about pairwise interaction between these measures for different datasets, unlike the use of projection in the construction of an instance space this method does not scale well to more problem features.

In Smith-Miles and Bowly (2015), the distribution of instances in the instance space is used to identify target points i.e. gaps in the space where there is a lack of datasets with those properties. For this, they simply used the Euclidean distance

between the defined target points and the problem feature vectors projected using the previously calculated principal components. This distance was then used as the fitness function for a GA to evolve new instances, though multiple strategies were used for how to set the target points. The difficulty of evolving instances with specific problem features depends on the flexibility of the generator, and the complexity of the problem features. For the graph colouring problem used in Smith-Miles and Bowly (2015), the best strategy found was to progressively move target points away from existing datasets, highlighting the difficulty with generating instances with specific properties.

Beyond the identification of algorithmic footprints, an instance space allows us to inspect how the problem feature values vary across the space for different datasets, either by their source (synthetic or real-world) or to identify differences in the underlying generating mechanism of synthetic datasets. The latter is particularly useful when it is not possible to explicitly generate data with certain properties, which is an issue with synthetic data generators for clustering. In the following section, we discuss some existing generators against which we later compare our proposed generator.

4.3.2 Synthetic cluster generators

Synthetic data is trivial to generate, as in the simplest case we can draw samples from simple distributions with little parameterization. The complexity occurs with the desire to generate synthetic data with various properties, including difficulties (e.g. noise) found in real-world data. In this section, we discuss a range of different methods for generating synthetic data for use in clustering. Note that we have selected a cross-section of work to highlight different approaches, rather than an exhaustive list of all works that have created synthetic data for clustering.

In the popular Python machine learning library *scikit-learn* (Pedregosa et al. 2011), there are several functions for generating data that sample from simple distributions or functions, three of which are illustrated in Figure 4.2. The first (Figure 4.2a) shows a toy dataset of two concentric circles (with some noise added to the samples) which, despite its simplicity and far-removal from real-world data, presents a clear and easily visualized challenge for compactness-based clustering algorithms. Figure 4.2b shows a function that provides an analogue for the connectedness property, which while useful provides parameterization only for the

(a) `Circles` (b) `Moons` (c) `Blobs`

Figure 4.2: An example dataset from each of three simple synthetic generator functions in *scikit-learn* (Pedregosa et al. 2011).

noise, and not shape or number of such clusters. In Figure 4.2c we can see a function that samples from a specified number of multivariate Gaussian distributions (four in this example). This function provides very limited control, as although the width of the distribution can be controlled, there are no covariances. As shown, two clusters have been placed almost directly overlapping (in the top-left), making it impossible to separate these and thus rendering any evaluation using the labels misleading. These functions/generators can help illustrate advantages or disadvantages of algorithms (as was shown in Figure 1.1), but they do not provide a comprehensive challenge, and only exhibit one such challenge for a given dataset.

The generator proposed in Qiu and Joe (2006a), henceforth *QJ*, uses a geometric framework for cluster placement. The measure of separation used was proposed in Qiu and Joe (2006b), named the "degree of separation". It provides a measure of the spatial separation between clusters, and the authors calculated values that *roughly* correspond to "close", "separated", and "well-separated" clusters. This value then imposes a minimum amount of separation that is accepted between any two clusters; the covariance matrices are iteratively scaled until this minimum separation is achieved. Although useful and geometrically interpretable, this provides a single perspective of cluster structure. Their generator does add an additional aspect of problem difficulty via the introduction of noise, in addition to explicit rotation of data points to increase the difficulty of identifying the number of clusters (Qiu and Joe 2006a). Although this generator has useful parameters to customize the difficulty of the generated problems, the generator is not easily extended to incorporate other cluster properties. An example dataset from this generator can be seen in Figure 4.3.

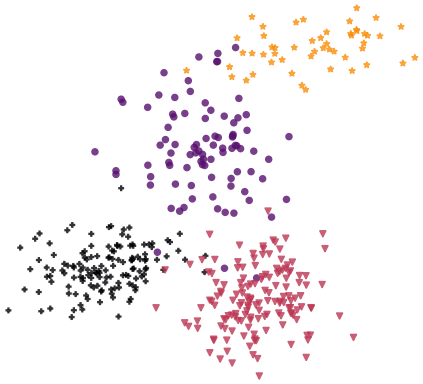
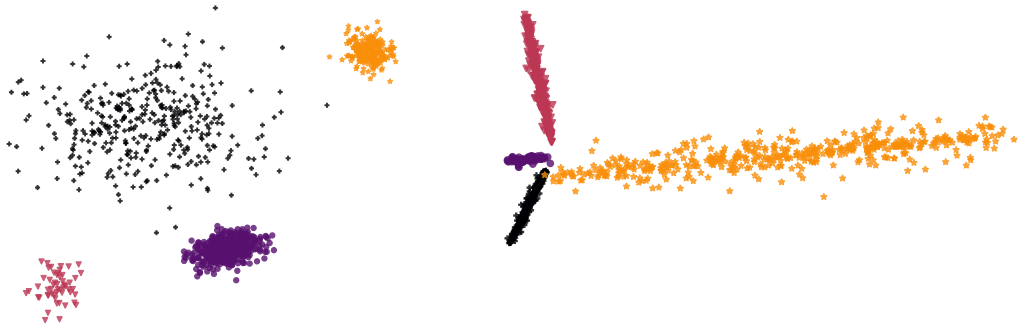


Figure 4.3: An example dataset from the *QJ* generator (Qiu and Joe 2006a).

A generator that has been used extensively was proposed in Handl and Knowles (2005b), henceforth referred to as *HK*. Owing to the difficulty of generating interesting (non-trivial) cluster structures in higher dimensions, two separate generators were proposed: "gaussian", and "ellipsoidal". The former uses a simple trial-and-error approach to place clusters in a pre-defined multi-dimensional space, while rejecting clusters that would result in overlap. Here, they define overlap as a data point's nearest neighbour belonging to a different cluster. The ellipsoidal generator uses a genetic algorithm to shift cluster positions to minimize the *overall deviation* (with a penalty for overlap) which, as defined in Handl and Knowles (2005a), is the sum of distances from each data point to its respective centroid. This generator does not consider systematic adjustment of cluster shape, nor does it consider a standardized measure of cluster validity (i.e. dimensionless, and thereby interpretable in its own right) as it simply aims to reduce this measure of compactness while penalizing overlap. Given the above, and the hard-coding of a large number of design choices, both parts of *HK*s generator are limited in their ability to produce instances with a specified level of difficulty and, therefore, in generating benchmarks that systematically test the performance of clustering methods with respect to particular aspects of problem difficulty. However, the explicit design of a generator for producing eccentric clusters in higher dimensions allows for cluster shapes that are otherwise difficult to generate (as seen in Section 6.2). Example datasets from these two generators can be seen in Figure 4.4.

Similar to Handl and Knowles (2005b), in Jing, Ng, and Huang (2007) the generator proposed was not the main focus of the work, rather it was created



(a) "Gaussian" (b) "Ellipsoidal" (projected to 2D via PCA)

Figure 4.4: Example datasets from the two synthetic generators proposed in Handl and Knowles (2005b).

specifically to test an aspect of the clustering approach they proposed (in this case, it was to explicitly create sparse data/clusters). For this, they generate clusters in a subset of the space, such that the feature values are either zero or random according to a "subspace ratio". An "overlap ratio" is also defined to determine to what degree the subspaces of clusters can overlap. This is similar to Zaït and Messatfa (1997), where clusters are generated in disjoint ranges of a continuous domain or (mostly) disjoint subsets of categorical features. In both of these works, clusters are generated in different domains such that they should be clearly separable, but no explicit measure of this is included in their generation.

At a similar time to our work (Shand et al. 2019), Gallagher (2019) proposed a method of generating clustering problem instances of varied difficulty for the experimental evaluation of continuous optimization algorithms. Despite a different target of evaluation (we directly target the evaluation of clustering algorithms), the desire to generate clustering problems with different characteristics is the same. This work similarly attempts to define key aspects of instance difficulty for clustering problems, though this is done through the perspective of fitness landscapes (where the objective is the same intra-cluster variance that K-Means uses, described in Equation 2.2). The identified characteristics that modify the difficulty of these instances are simple descriptors of the data, namely the number of clusters (K), dimensions (D), and data points (N).

The proposed instance generation focuses entirely on *compact* clusters, however, and thus provides little utility in the generation of problems useful for

non-compactness-based algorithms. In terms of generating mechanism, similar to Macià, Orriols-Puig, and Bernadó-Mansilla (2010) and Macià and Bernadó-Mansilla (2014), Gallagher (2019) generates data through modification of the existing examples. A subset of the data is selected that maximally degrades the performance of an EA (measured by the time to converge). Such a subset may no longer represent what the clusters *should* be, potentially providing misleading results. As a result of this method, no explicit generator is provided for the synthetic data in Gallagher (2019), and thus will not be included alongside the *QJ* & *HK* generators in our later work.

4.3.3 Evolving instances

As illustrated in the previous section, instances can have a variety of generating mechanisms that vary greatly in complexity. If we have measures that give an indication as to the properties of our generated instance, then the use of heuristics or optimization such that we can guide the generation of our instances to certain values of these properties is desirable. While certain combinations of properties may be impossible, if there is a trade-off between these properties then a multi-objective optimization approach (Section 3.1) is far more useful than a random generation mechanism. In this section, we explore existing work that uses EAs as the generating mechanism to create instances (apart from the previously discussed *HK* generator).

In the previous section, we discussed a method of generating data that involved modifying existing datasets, rather than constructing them from scratch. A major issue with such methods is that the meaningfulness of difficulty is lost, as the removal of data points may result in a change of what the labels *should* be, which would not be reflected in subsequent evaluation. Macià, Orriols-Puig, and Bernadó-Mansilla (2010) and Macià and Bernadó-Mansilla (2014) use NSGA-II to select a subset of the data to optimize complexity measures (such as the fraction of data points on the class boundary). The constraints used (such as a minimum number of instances and maintaining a class balance) can alleviate the issue of distorting what the classes *should* be, but as these are user-defined parameters this is not guaranteed. Macià and Bernadó-Mansilla (2014) apply this method to datasets from the UCI repository (Dheeru and Karra Taniskidou 2017) to construct samples from these datasets that exhibit different measures of complexity, which are then visualized using a "complexity space" that is similar

to the aforementioned instance space (Section 4.3.1) we utilize in this thesis.

Hemert (2006) uses an EA to generate datasets for different combinatorial optimization problems, using the search effort (such as the number of constraint checks) of a pre-defined solver as the fitness and thus indicator of difficulty. As a result, only one perspective of difficulty is taken into account and other problem features are not explicitly incorporated. By measuring the search effort for a particular algorithm without explicit measurement or consideration of general problem properties, the generated datasets have potentially limited transfer to other algorithms. The comparison of search effort between algorithms can offer insights but, the lack of problem features obscures the *causal* reason behind such differences.

Utilizing yet another EA paradigm, Lensen, Xue, and Zhang (2018) use genetic programming (GP) to evolve datasets for feature selection. Their motivation for this is similar in concept to ours for clustering; that is, the need to construct a dataset that has defined properties which are difficult to measure. In their case, identifying which features of a dataset are non-linearly redundant (i.e. share information with existing features, but the relationship is not a simple linear correlation) is difficult, thus warranting the generation of datasets with these properties. In their work, the datasets themselves are not evolved from nothing — existing datasets are used as the initial features, from which additional features are constructed. While their approach was both useful and provided potentially more informative features for downstream analysis, there was no overall analysis of the diversity of the datasets produced.

As mentioned in Section 4.3.1, Smith-Miles and Bowly (2015) created an instance space for (vertex) graph colouring, a challenging optimization task where vertices on a graph need to be assigned a colour such that no two vertices sharing an edge have the same colour. In this work, evolution was used as a means of generating datasets with more explicit properties. Using the instance space, they identified target points within this space that were missing instances and as such attempted to generate instances at these points. They used a GA to evolve individuals (graphs), experimenting with different strategies for the selection of target points in the instance space. Multiple strategies were used to generate datasets that fill the instance space (by modifying either the fitness function or how target points in the instance space are identified). An interesting strategy creates a spread of target points close to existing instances, and iteratively moves

these points during optimization to help guide the search and accelerate convergence. As a result, they generate datasets with a significantly higher spread across the instance space (note, however, that there is a small difference in algorithm performance with the new instances). Nevertheless, there is clear utility in not only generating instances with properties that have been identified as absent from existing data, but directly using the proximity of current individuals to those in the instance space *during* optimization to drive this evolution. This is particularly useful if the original optimization (both the fitness and constraints) are loosely associated with the problem features.

4.3.4 What makes a cluster "difficult"?

We have previously discussed the need for a diverse range of datasets in order to gain algorithmic insights. Generally, clustering algorithms optimize using a single measure of (internal) cluster quality (Section 2.5.1). In Section 2.3 we discussed some of the different properties of clusters that need to be captured by these indices. These are, in essence, a non-exhaustive set of properties that can make the cluster structure clearer or harder to discover for different algorithms. Evaluating the difficulty (and thus diversity) of datasets requires in part the use of these indices, which can lead to the introduction of biases towards algorithms that utilize the same or similar indices. As these biases can be difficult to identify, the use of multiple, carefully selected (i.e. complementary) internal indices and other properties is required to ascertain the difficulty of any clustering problem with as broad a perspective as possible.

Examples of different properties and the difficulty they pose for different algorithms can be seen in Figure 1.1, with specific reference to the three eccentric clusters that have been reproduced here in Figure 4.5 for ease. Referring to the properties defined in Handl, Knowles, and Kell (2005), these clusters have high connectedness and low compactness, in addition to the fact that they are close to overlapping with each other (low spatial separation). As different clustering algorithms are biased towards different cluster structures, this example has properties that can vary the difficulty for algorithms in different ways. Referring to the cluster assignments in Figure 4.5, single-linkage struggles to separate the clusters as they are close enough it looks like one big continuous cluster (beyond the 'outliers' in the top-left and bottom-right), and K-Means++ has difficulties due to the eccentricities of the clusters. Owing to their generation from a Gaussian

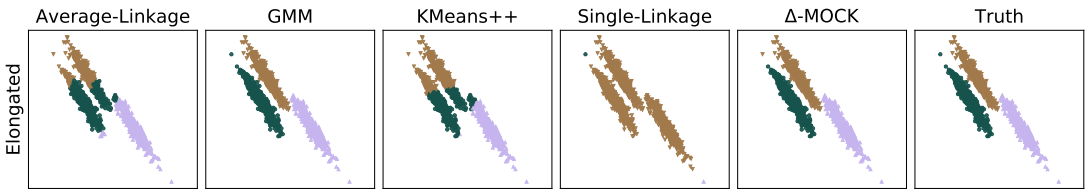


Figure 4.5: Varied behaviour/performance of different clustering algorithms on three eccentric Gaussian clusters (subset of Figure 1.1).

distribution, GMM finds the exact assignment.

Understanding the different facets of difficulty can allow for these to be used in the generation of synthetic clusters. As discussed, increasing the difficulty according to a single property would result in a non-uniform response from a portfolio of clustering algorithms. Generating synthetic data for clustering would require a flexible tool where the various levers of difficulty can be pulled (ideally) independently if we are to gain algorithmic insights. In Chapter 5, we take our first step to creating such a generator.

4.4 Summary

In this chapter we explored the role of synthetic data in experimentation, and how it is useful in tackling the algorithm selection problem. We reviewed different existing synthetic data generators specific to clustering, and the challenges that such a generator must face to be broadly useful. Combined with the generative power of EAs (Chapter 3) and the multi-objective, multi-faceted nature of difficulty for cluster analysis (Chapter 2), in the next chapter we introduce an EA that evolves cluster structure to create synthetic data of varied difficulty for different clustering algorithms.

Chapter 5

Evolving Difficult Synthetic Clusters I

Following from Chapter 2 and Chapter 4, there is a clear need for a comprehensive synthetic data generator that can be used in clustering. Such a generator needs to consider different properties that pose different challenges for different algorithms, while having the flexibility to parameterize and optimize multiple properties in order to produce a diverse set of datasets. As such, EAs (Chapter 3) provide a great approach for this task. In this chapter, we introduce our synthetic data generator HAWKS, first beginning with a description of its components and the design decisions behind them (Section 5.1). This is followed in Section 5.2 by an evaluation of HAWKS, particularly in comparison with other popular generators. Finally, from these experiments we identify several areas of improvement for HAWKS in Section 5.3.

5.1 HAWKS

In this section, we introduce and discuss HAWKS[1], a synthetic data generator. Some of the content in this chapter is adapted from Shand et al. (2019). Details on where to find and use HAWKS can be found in Section A.1.

[1]The name is derived from the surnames of the people involved: **H**andl, **A**llmendinger, **W**ebb, **K**eane, and **S**hand. Note that the order is irrespective of contribution, it's just the best acronym I could think of.

Algorithm 5.1: HAWKS

input : N, K, D, $G_{\max}$, $|\mathcal{P}|$, P_f, s_{target}, $C_{\min}$, β_1, β_2
output: Population of datasets ($\mathcal{P}$)

```
1 P ← initialization(N, K, D, |P|, C_min, β1, β2)        // Section 5.1.1
2 evaluation(P, s_target)                                // Section 5.1.2, Section 5.1.3
3 for gen ← 1 to G_max do
4 |   P' ← parental_selection(P)                          // Section 5.1.5
5 |   P'' ← genetic_operators(P')                         // Section 5.1.4
6 |   evaluation(P'', s_target)
7 |   P ← stochastic_ranking(P ∪ P'', P_f)                // Section 5.1.5
8 end
```

HAWKS uses an EA (evolutionary algorithm) to generate and optimize a population of datasets, subject to constraints that encourage certain cluster properties. Using an EA provides a flexible, modular framework that enables the use of many different components, which is necessary for a broadly capable synthetic data generator. The stochasticity of EAs is also favourable in this context, as it allows for the generation of multiple datasets using the same set of parameters, potentially creating a number of diverse datasets (either in the same or in different initializations).

In Algorithm 5.1, we provide a general overview of HAWKS to provide context for the remainder of this section. Overall, a generic EA framework is used, but owing to the complexity of the design decisions made in creating a generator to evolve clustering datasets, these are discussed in their own sections.

In Section 5.1.1, we discuss how we parameterize and encode a dataset as an individual, as well as how an initial population of individuals is created. The evaluation of an individual, encompassing its fitness and constraints, is covered in Sections 5.1.2 and 5.1.3 respectively, where we discuss the pitfalls and complexities of defining the fitness of a dataset, and how constraints can be used to add additional cluster properties without disrupting convergence. Section 5.1.4 discusses how to design meaningful genetic operators that can perturb clusters in a way that directly affects the fitness while ensuring that HAWKS retains flexibility. Finally, in Section 5.1.5 we discuss how to select which individuals should be parents, and which should be selected for the next generation and thus drive the evolution forward.

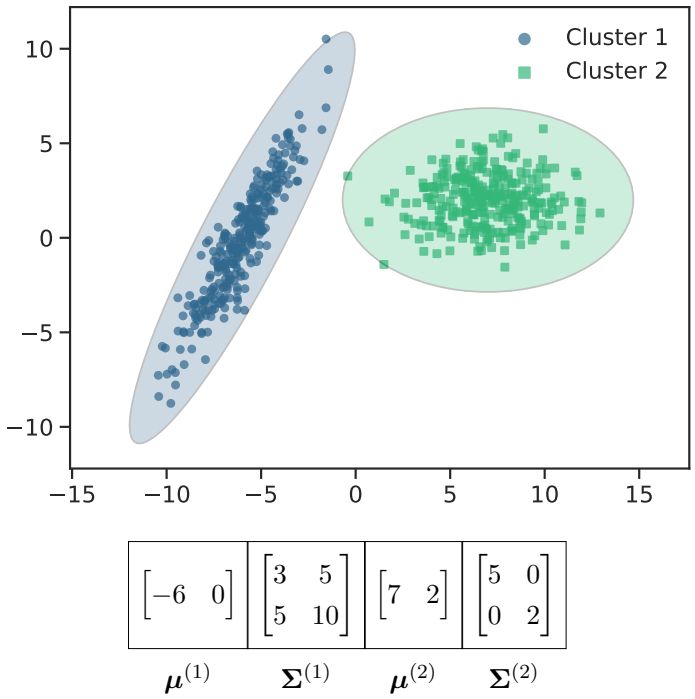


Figure 5.1: Example representation of a two-cluster problem instance.

The input parameters for HAWKS in Algorithm 5.1 will be explained through this chapter in their relevant sections. For each experiment in this thesis, the full set of parameters for HAWKS are provided in a configuration file. Please refer to Section A.1.1 for further details on this and more generally how to replicate our experiments.

5.1.1 Encoding a dataset

As discussed in Section 3.2.1, the representation (or encoding) used is vital to the success of an EA. We use a simple encoding scheme to represent the dataset, where each cluster is a (multivariate) Gaussian distribution encoded by its mean ($\boldsymbol{\mu}$) and covariance ($\boldsymbol{\Sigma}$). Treating each cluster as a "gene" of a $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$-pair, the genotype is of length K genes. Figure 5.1 shows an example of this representation and the corresponding two-cluster dataset, where the actual points sampled from the cluster and the corresponding distribution (to 3 standard deviations) are shown. In the genotype underneath, each mean and covariance contains D and $D(D+1)/2$ (independent) variables respectively, which can be important when considering the genetic operators.

As the data points themselves are sampled from the encoded distributions, this representation has some non-standard properties that are important to note. Although the details of the fitness function will be detailed in Section 5.1.2, if we assume it is some quantity representing cluster structure then it's clear that a rotation of the dataset will significantly change the genotype, but the fitness will be unchanged. Thus, as multiple genotypes can have the same fitness our representation is synonymously redundant (defined in Section 3.2.1).

Further, it's clear that stochastic sampling of data points from the Gaussian distributions can potentially result in different fitnesses for the same genotype, adding further complication to the evolution. To alleviate this issue and better guide the optimization, the samplings from each distribution are fixed (for each individual cluster via a constant random seed). As a result, a rotation of the covariance matrix is synonymous with a rotation of the sampled data points. To assess the sensitivity of our encoding to different samplings, we perform a sensitivity analysis in Section A.2, where it was clear that the individuals varied (in terms of their fitness) far more across a population than for different samplings of the same genotype.

Then, we need to determine the number of points to sample from each distribution. The total number of data points (N) for the datasets is user-defined, and the size of each cluster is set during the initialization and remains fixed throughout the optimization across all individuals (rather than encoded). This ensures that there is no negative interaction with the fitness, which will be explained further in Section 5.1.2. We discuss the initialization of the cluster sizes and the individuals themselves in the following section.

Initialization

Algorithm 5.2 provides pseudocode for the initialization (referred to on line 1 in Algorithm 5.1) that creates the initial population of datasets, denoted $\mathcal{P}$ (and thus $|\mathcal{P}|$ the number of individuals in the population, which we use in place of the traditional symbol μ to avoid ambiguity). This will provide further details on how we generate random clusters, including the task of generating random (valid) covariance matrices to create a diverse set of clusters.

Although the sizes of the clusters are fixed during the optimization, HAWKS provides a degree of control over the sizes of the generated clusters as this can challenge a clustering algorithm's robustness to dealing with differences in density

Algorithm 5.2: HAWKS initialization

1 **Function** `initialization`(N, K, D, $|\mathcal{P}|$, C_{min}, β_1, β_2)
2 `generate_cluster_sizes`(N, C_{min})
3 $\mathcal{P} \leftarrow \varnothing$
4 **for** $i \leftarrow 1$ *to* $|\mathcal{P}|$ **do**
5 $\mathbf{x}_i \leftarrow \varnothing$
6 **for** $k \leftarrow 1$ *to* K **do**
7 $\boldsymbol{\mu}^{(k)} \leftarrow \mathcal{U}[0, \beta_1]^D$
8 $\widetilde{\boldsymbol{\Sigma}}^{(k)} \leftarrow \text{diag}(\mathcal{U}[0, \beta_2]^D)$
9 $\boldsymbol{R}^{(k)}, \boldsymbol{S}^{(k)} \leftarrow$ `sample_matrices()`
10 $\boldsymbol{\Sigma}^{(k)} \leftarrow \boldsymbol{R}^{(k)} \cdot \boldsymbol{S}^{(k)} \cdot \widetilde{\boldsymbol{\Sigma}}^{(k)} \cdot \boldsymbol{R}^{(k)\intercal}$
11 $\mathbf{x}_i \leftarrow \mathbf{x}_i \cup \{\boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)}\}$
12 **end**
13 $\mathcal{P} \leftarrow \mathcal{P} \cup \mathbf{x}_i$
14 **end**
15 **return** $\mathcal{P}$

(as illustrated in Figure 2.5a). The clusters can be forced to be equally sized, but otherwise they are generated randomly such that they sum to the pre-specified N, with an optional minimum cluster size (C_{min}) imposed. For completeness, and owing to its non-triviality, the method of generating random cluster sizes with a minimum size that also sum to a defined value is provided in Section A.3. This is the procedure referred to in Algorithm 5.2 (line 2). As an example, if we specify $N = 500$ and $K = 5$, and the cluster sizes were randomly generated as $\{|C_k| : C_k \in \mathcal{C}\} = \{34, 191, 95, 58, 122\}$, then every individual will have clusters of those sizes in every generation, forcing the differences to be in the parameters ($\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$) of the clusters themselves.

To initialize the individuals, we need to randomly generate the means and the covariances. As shown on line 7 in Algorithm 5.2, the kth mean, $\boldsymbol{\mu}^{(k)}$, is generated as follows:

$$\boldsymbol{\mu}^{(k)} \sim \mathcal{U}[0, \beta_1]^D, \tag{5.1}$$

where $\mathcal{U}$ is the uniform distribution, and β_1 specifies the upper bound that the mean values can be sampled from. Thus, the mean is simply sampled as a random point in a D-dimensional hypercube.

The covariance matrix in HAWKS is constructed from several different components. The kth covariance matrix, $\boldsymbol{\Sigma}^{(k)}$, is constructed as follows:

$$\boldsymbol{\Sigma}^{(k)} = \boldsymbol{R}^{(k)}\boldsymbol{S}^{(k)}\widetilde{\boldsymbol{\Sigma}}^{(k)}\boldsymbol{R}^{(k)\intercal}, \tag{5.2}$$

where $\boldsymbol{R}^{(k)}$ and $\boldsymbol{S}^{(k)}$ are the kth rotation and scaling matrices respectively, and $\widetilde{\boldsymbol{\Sigma}}^{(k)}$ is the kth axis-aligned covariance matrix (i.e. a diagonal matrix that consists of only the variances). Decomposing the covariance matrix allows us to modify the rotation and scaling matrices separately, which is important for the mutation operators discussed in Section 5.1.4, as well as allowing for more intuitive initialization parameters.

The axis-aligned covariance matrix for the kth cluster is sampled as follows:

$$\widetilde{\boldsymbol{\Sigma}}^{(k)} \sim \text{diag}(\mathcal{U}[0, \beta_2]^D), \tag{5.3}$$

where β_2 is a parameter that controls the magnitude of the initial variances and $\text{diag}(\cdot)$ of some D-dimensional vector produces a $D \times D$ diagonal matrix whose entries are the elements of that vector.

The rotation matrix, $\boldsymbol{R}$, modifies the axis-aligned covariance matrix such that the resulting covariance matrix is non-diagonal i.e. it has covariances. Visually, the difference can be seen in Figure 5.1 where the second cluster has no covariance. The initial rotation matrix is created by drawing a random orthogonal matrix from the Haar distribution (Stewart 1980). This ensures that the covariance matrix after rotation is valid (i.e. positive semi-definite). This matrix is generated using `scipy`[2] (Virtanen et al. 2019). The initial scaling matrix is the D-dimensional identity matrix, $\mathcal{I}_D$, as this is not needed for the initialization (but is useful later in the mutation).

The initial sampling range of the means and covariances (controlled by β_1 and β_2) influence the likelihood of whether the clusters are initialized as overlapping or not. This will have a direct influence on convergence, depending on what the final desired cluster structure is and their starting positions. Additionally, higher values for β_2 increase the chance of having more eccentric clusters, which can pose difficulties for compactness-based clustering algorithms (such as K-Means). Further guidance and considerations on these parameters will be provided later in Section 5.1.4.

[2]The `scipy.stats.special_ortho_group` function is used.

5.1.2 What is the fitness of a dataset?

As previously mentioned, the fitness of the data should quantify in some way the cluster structure and by extension the potential difficulty. However, as discussed in Chapter 4, identifying a useful and diverse set of problem features to measure the difficulty of a dataset is a complex issue which is further complicated when used in clustering. For our first iteration of HAWKS, we want a simple, single-objective algorithm that incorporates several elements of difficulty. To control the dataset difficulty, we optimize towards a user-defined value of difficulty, which can be represented through e.g. cluster validity indices and/or other measures of different cluster properties (Section 2.3).

Directly maximizing or minimizing such measures would generate datasets of no use in gaining insights into algorithms. If we use the intra-cluster variance (defined in Equation 2.2) as an example, a dataset that is evolved to minimize this would result in trivially separable clusters (by placing each point in its own cluster) that pose no challenge to any competent algorithm. Conversely, maximizing the intra-cluster variance would attempt to move data points in the same cluster as far away from their centroid as possible, creating a dataset that has no actual cluster structure. Without other measures, such as the separation between clusters, the overall cluster structure would be poor/non-existent.

While optimizing towards a user-defined value can avoid these extremes, the measure used must have an interpretable target value to provide intuition for the user. As a result, it should be bounded, dimensionless, and easy to understand. Additionally, a more complex cluster validity index that takes into account compactness and inter-cluster separation would help to provide a more holistic view of the "fitness" of the dataset.

As such, we selected the silhouette width (see Section 2.5.1 and Equation 2.4 for details) as a proxy for the difficulty, where the user provides a target silhouette width (s_{target}) that is optimized towards. The extensive study by Arbelaitz et al. (2013) indicated that the silhouette width was notably one of the most useful indices out of 30 that were studied (though they could not conclude it was superior). Importantly for HAWKS, it provides a dimensionless (and thus comparable across datasets of the same dimensionality) measure of average compactness and separation in the range $[-1, 1]$, and is therefore more interpretable.

Our fitness function is thus formulated to minimize the absolute difference

between s_{target} and s_{all}, defined as:

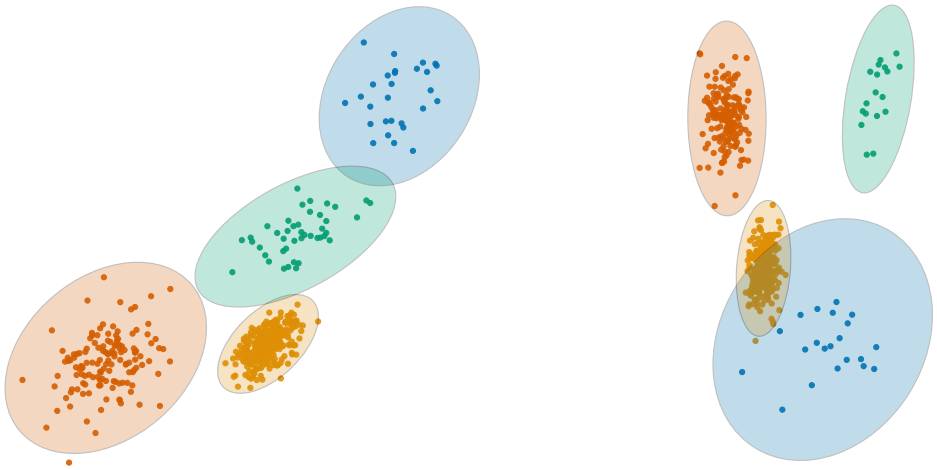
$$\min\ f(X) \equiv \min\ |s_{\text{target}} - s_{all}| \tag{5.4}$$

where $s_{all} = \frac{1}{N}\sum_{\mathbf{x}\in X} s(\mathbf{x})$, and X is the set of data points sampled from the K encoded Gaussian distributions.

Before we discuss the limitations of the silhouette width, which are important to consider in the design of HAWKS, it should be noted that (as discussed in Section 2.5.1) all indices have some drawbacks that need to be considered. One of the main limitations of the silhouette width is its computational complexity, $O(DN^2)$, which has made its use as an optimization objective problematic (Vendramin, Campello, and Hruschka 2010). However, as the data points in our datasets/individuals change only due to the genetic operators, we are able to reduce this computation to only the clusters that change, thereby permitting the re-calculation of only a subset of pairwise distances. As such, the above complexity is the worst-case that only occurs when every cluster in a dataset has been modified. Even without this improvement, however, the generation of datasets is rarely a time-constrained environment, making this a less important consideration than when used in clustering algorithms themselves.

Another issue with the silhouette width, that is particularly pertinent for optimization, is the fact that it is an average across all points. This fact presents two primary issues in need of consideration when being used in an EA: for lower N, each individual data point has a relatively higher contribution to s_{all} such that outliers or samples far from the distribution's mean may result in a misrepresentative s_{all} value; and, a dataset with equally well-separated clusters may have the same s_{all} as a dataset that contains both completely overlapping and *very* well-separated clusters as long as the vast majority of actual data points are separated. As such we can have a 'deceptive' (higher than expected) silhouette width, an example of which was generated with HAWKS and is shown in Figure 5.2. This behaviour also motivated the need to fix the cluster sizes throughout the optimization, otherwise changing the silhouette width can easily be achieved by changing of cluster sizes rather than their location,the latter of which is of far greater importance towards the goal of generating interesting datasets.

As previously discussed (Section 2.5.1), the limited perspective of 'difficulty' that cluster validity indices such as the silhouette width provide leads to the natural desire to simultaneously incorporate information from multiple such indices.



(a) Separated (b) Deceptive

Figure 5.2: Two example datasets (generated by HAWKS), both with $s_{all} = 0.9$. As this value is an average we can optimize to a high silhouette width by placing larger clusters further away, even if small clusters end up overlapping. The points and 3 standard deviations are shown for each cluster.

Therein follows the question why we are using a single objective. First, for a useful set of complementary indices, we may require many such indices to the point that we create a difficult *many*-objective problem. Second, the non-trivial interaction between these measures during the search drastically complicates both the evolution and the ability of the user to specify the difficulty. Overall, the silhouette width provides a broadly useful indicator of cluster structure that is comparable between datasets in a population, and in a range that can be easily understood by users. We can further modulate this difficulty however, through the introduction of constraints that would not be useful in isolation as an objective.

5.1.3 Augmenting difficulty through constraints

To counter the limited perspective of difficulty provided by the silhouette width, two constraints are added to incorporate additional cluster properties.

Elongation

As previously discussed in Section 4.3.4, the elongation of a cluster is an important component of difficulty for different clustering algorithms. As HAWKS uses

Gaussian (and therefore ellipsoidal) clusters, we refer to elongation more specifically as eccentricity. To quantify and control this eccentricity, we can calculate the ratio between the largest and smallest eigenvalues of the covariance matrices. This ratio is calculated for each cluster in an individual, from which the maximum is then taken, as follows:

$$\lambda^{ratio} = \max_{\forall\, k \in \{1,\dots,K\}} \frac{|\lambda_{\max}(\boldsymbol{\Sigma}^{(k)})|}{|\lambda_{\min}(\boldsymbol{\Sigma}^{(k)})|}, \tag{5.5}$$

where $\lambda_{\max}(\boldsymbol{\Sigma}^{(k)})$ and $\lambda_{\min}(\boldsymbol{\Sigma}^{(k)})$ are the maximal and minimal eigenvalues of $\boldsymbol{\Sigma}^{(k)}$ respectively.[3]

Controlling the amount of eccentricity that leads to an infeasible solution can allow for the creation (or prevention) of elongated clusters, depending on whether an upper or lower bound is set to calculate the constraint violation. This can be used to introduce different challenges for clustering algorithms by having highly compact (or elongated) clusters. This ratio has a lower bound of 1, which indicates perfect sphericity, and no upper bound.

Overlap

Noise is prevalent in real-world data, and one particularly challenging consequence of this is when clusters have a degree of overlap which can further complicate identification of cluster membership. The robustness of algorithms to overlap varies significantly, and therefore having a degree of control over this property helps provide a wider diversity of cluster structures that can be generated. By formulating this as a constraint, we can either penalize or encourage a pre-defined amount of overlap.

The concept of overlap itself is not purely objective. For example, Fränti and Sieranoja (2018) use a measure that determines a point as overlapping if the distance to its own centroid is higher than the distance to a point in another cluster. In this formulation, highly eccentric clusters may be seen to "overlap" even if spatially separated (and thus visually be separated, which is a more intuitive understanding of overlap). We use the same formulation as Handl and Knowles (2005b), where a data point is defined as overlapping if the point's nearest neighbour is a member of a different cluster than itself. This provides a more

[3]Note that λ^{ratio} is equivalent to the maximum condition number across all K covariance matrices.

intuitive measure that is less biased towards spherical clusters, and has the additional advantage of discouraging the aforementioned 'deceptive' silhouette width (Figure 5.2).

The overall overlap is calculated as the percentage of points that match this criterion, and is scaled into the range $[0, 1]$. It is important to note, however, that for two perfectly super-imposed Gaussian clusters (C_i and C_j) with identical $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, and sampling density (i.e. $|C_i| = |C_j|$), the expected overlap is 0.5 and not 1, as a given data point has a 50% chance of being in either cluster. Values higher than this can be achieved with multiple overlapping clusters, or overlapping clusters of different densities.

Before we formally define the overlap, we first need to define notation for a data point's nearest neighbour. Let $n_{\mathbf{x}} : \{1, \ldots, N\} \to X$ be a function such that $n_{\mathbf{x}}(j)$ is the jth nearest neighbour of data point $\mathbf{x}$. This function can be viewed as indexing into a re-ordered dataset $(\mathbf{x}'_i)_{i=1}^N$ such that $d(\mathbf{x}, \mathbf{x}'_1) \leq \cdots \leq d(\mathbf{x}, \mathbf{x}'_N)$. The first nearest neighbour is the data point itself, i.e. $n_{\mathbf{x}}(1) = \mathbf{x}$.

We define the overlap formally as:

$$overlap = 1 - \frac{1}{N} \sum_{\mathbf{x} \in X} \mathbb{1}_{C_k}(n_{\mathbf{x}}(2)) \tag{5.6}$$

where C_k is the cluster that data point $\mathbf{x}$ belongs to, and $\mathbb{1}$ is the indicator function defined as:

$$\mathbb{1}_{C_k}(n_{\mathbf{x}}(2)) := \begin{cases} 1, & \text{if } n_{\mathbf{x}}(2) \in C_k \\ 0, & \text{if } n_{\mathbf{x}}(2) \notin C_k \end{cases}$$

For the remainder of this thesis, we refer to this constraint explicitly as *overlap*, to disambiguate from the general concept of overlap which we also heavily refer to.

Constraint handling

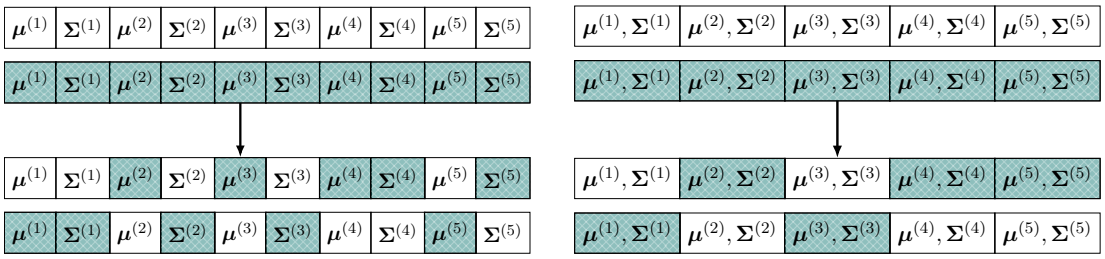
As discussed in Section 3.3, the general purpose of constraints is to penalize solutions with either undesirable (soft constraints) or unacceptable (hard constraints) constraint values for a given genotype. For constraint-handling techniques that seek to balance exploration of the infeasible region against exploitation of the feasible region, feasibility is at least preferred, if not required. Our constraints can

conflict with the fitness such that it is impossible to minimize the fitness with no constraint violation. This can potentially create many infeasible solutions (that are still useful or even desirable) during evolution, as would be the case if we were to generate datasets with e.g. varying degrees of *overlap* for a given silhouette width. For certain sets of parameters (such as a high s_{target} and high *overlap*) there may be no feasible solutions, which can disrupt the guided process of the search if not appropriately handled.

To tackle this, we use stochastic ranking (Runarsson and Yao 2000) to handle the trade-off between satisfying the constraints and the objective function. For a full explanation of the stochastic ranking, see Section 3.3.2. In brief, the P_f parameter in stochastic ranking specifies the probability that two (infeasible) solutions are compared using the objective function rather than their constraint penalties. At low P_f values we favour lower constraint violation, and at higher values we favour greater minimization of $|s_{\text{target}} - s_{all}|$.

For the constraints we have in HAWKS, solutions that violate the constraints (to a degree) are acceptable and may even be desired depending on the *intended use* of the generated datasets. Particularly in the aforementioned scenario where both a high *overlap* and a high silhouette width is desired, having a balanced trade-off between these non-mutually satisfiable properties allows for the generation of varied and potentially interesting datasets. In the absence of exploring this trade-off through techniques available in multi-objective optimization (by treating the constraints as objectives — discussed in Section 3.3.2), the stochastic ranking allows us to directly specify a preference for this trade-off by adjusting P_f. For each of our two constraints, a lower or upper threshold is set, where going outside of the permitted range will incur a penalty of the squared difference between the threshold and the actual value obtained (Runarsson and Yao 2000). The utility of using the stochastic ranking to embed a preference will be investigated in Section 5.2.3.

Having calculated the fitness of an individual, variation is needed in order to generate new datasets and drive the evolution forward. In the next section, we look at how to design genetic operators that are meaningful in the context of clustering.



(a) Components (b) Whole clusters

Figure 5.3: Uniform crossover, swapping either the individual components (a) or the ($\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)-pair (b).

5.1.4 Perturbing a dataset

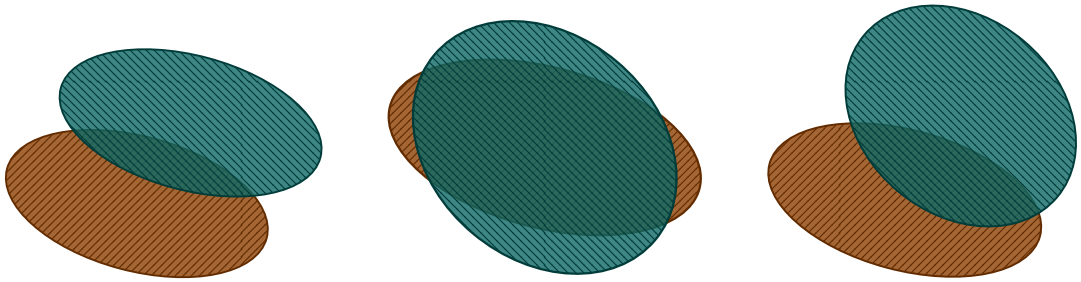
A core part of an EA is its genetic operators, providing a source of perturbation to help the search. As each ($\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)-pair in our representation is a cluster, we can design our operators to modify individual clusters (rather than operating purely on decision variables), which has an intuitive and geometric advantage that enables us to directly and meaningfully perturb cluster *structure*, which is key to the generation of diverse and difficult datasets.

Crossover

We use standard uniform crossover (see Section 3.2.2 for background information), but considered at two resolution levels as illustrated in Figure 5.3. In Figure 5.3a, each $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is treated independently for crossover, whereas in Figure 5.3b the ($\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)-pair is considered as a single unit to swap between individuals. Intuitively, it stands to reason that as the fitness measures the intrinsic spatial structure of the dataset, there is a united contribution of the mean and covariance to this measure, and breaking up this single unit may be detrimental or even misleading to the search. Preliminary tests (not shown here) did not show a significant difference between these two operators, however.

Mutation

For the mutation, we wanted to ensure that there was no bias towards particular shapes or structures while maintaining significant random perturbation to permit sufficient exploration. For this, mutation of the mean and the covariance is handled by separate operators.



(a) Mean (b) Covariance (c) Mean and covariance

Figure 5.4: Illustration of HAWKS' mutation operator for the mean, covariance, and both combined.

To mutate the mean, we simply sample a new mean, denoted $\boldsymbol{\mu}'^{(k)}$, from a normal distribution around the current mean:

$$\boldsymbol{\mu}'^{(k)} \sim \mathcal{N}(\boldsymbol{\mu}^{(k)}, \sigma^2) \tag{5.7}$$

where σ^2 is the variance of the normal distribution. Thus, we effectively shift the mean of the cluster in a random direction, as illustrated in Figure 5.4a. To avoid excessive mutation rates, this new mean is actually sampled from each dimension individually (from a univariate Gaussian) with probability $\frac{1}{D}$, so that the mean is shifted in a subset of the possible dimensions. As there are more directions that can move a cluster away from the other clusters than towards in higher dimensions, this helps slow an increasing drift of the silhouette width (s_{all}) after multiple mutations.[4]

To mutate the covariance, we revisit the scaling and rotation matrices discussed in Section 5.1.1. Fundamentally, the scaling matrix acts to modify the eigenvalues (the cluster's eccentricity) of the covariance matrix, while the rotation matrix modifies the eigenvectors (the cluster's orientation).

For the scaling matrix $\boldsymbol{S}$, we found that additive or multiplicative scaling led to a biased distortion of cluster shape after repeated applications, reducing the diversity of the resulting datasets. To alleviate this, we ensure that the determinant of the covariance matrix remains the same after applying the scaling matrix. For this, let z_i be the elements of a vector sampled from a Dirichlet

[4]This issue is later addressed in Section 6.1.

distribution, then:

$$\begin{aligned} \sum_{i}^{D} z_i &= 1 \\ \Rightarrow \sum_{i}^{D} \left(z_i - \frac{1}{D} \right) &= 0 \\ \Rightarrow \exp\left(\sum_{i}^{D} \left(z_i - \frac{1}{D} \right) \right) &= 1 \\ \Rightarrow \prod_{i}^{D} \exp\left(z_i - \frac{1}{D} \right) &= 1. \end{aligned}$$

The determinant of a diagonal matrix is the product of the values on the diagonal, thus the scaling matrix ($\boldsymbol{S} = \exp(z_i - \frac{1}{D})$) has determinant 1.

The rotation matrix, $\boldsymbol{R}$, is created by drawing a random orthogonal matrix from the Haar distribution as described in Section 5.1.1. Here, however, the rotation matrix is raised to a fractional power (defaulted to 0.3) to avoid a complete reorientation of the cluster, helping to maintain the incremental, guided process of the optimization. The rotation and scaling matrices then perturb $\boldsymbol{\Sigma}$ as shown in Equation 5.2.

By way of analogy, $\boldsymbol{R}$ and $\boldsymbol{S}$ act to rotate a balloon and apply pressure to the principal semi-axes respectively, thereby changing the shape while maintaining the volume. The effect of this is shown in Figure 5.4b. The combined effect of both the mean and covariance operators is shown in Figure 5.4c, which shows both a change in location and shape of the cluster.

For the mutation probability, we use standard length-based mutation probabilities separately for the mean and covariance, i.e. the mean and covariance each have a $\frac{1}{K}$ probability of mutating. Despite the difference in the number of decision variables between the means ($K \times D$) and covariances ($K \times D(D+1)/2$), it is more meaningful to consider mutation at the $\boldsymbol{\mu}$ or $\boldsymbol{\Sigma}$ level as opposed to isolated decision variables.

In Section 5.1.1, we discussed the role of the β_1 and β_2 in the convergence of HAWKS as they determine the initial location and shape (respectively) of the clusters. As the dimensionality increases, there are more directions in which clusters can move away from each other than towards one another, which is problematic for the mean mutation operator (Equation 5.7). As a result, throughout this chapter we choose to initialize the clusters closer together, which makes the

evolution easier by perturbing these clusters until they are as compact as needed for the given s_{target} and constraints. If they begin very far apart, as D increases it is less likely to make perturbations that bring them closer together. Initial experiments (not shown here) with well-separated clusters resulted in much slower convergence when optimizing to lower silhouette width values.

The absolute values of β_1 and β_2 are less important than their relative values, particularly as the silhouette width produces a ratio (and thus the magnitude of the distances is irrelevant). The only caveat is that the mutation step-size (the variance of the Gaussian in the mean mutation operator, σ^2) is affected by the magnitude of these distances. Preliminary experiments (not shown) indicated setting these three parameters to 1 does not disrupt convergence to s_{target}.

5.1.5 Selection a dataset

An important part of an EA is the selection of both the parents and particularly which individuals survive to the next generation. In this section, we discuss the parental and environmental selection schemes (lines 4 and 7 respectively in Algorithm 5.1) that HAWKS uses to drive the evolution of our datasets forward.

Parental selection Binary tournament is used for the parental selection (background information on this method is given in Section 3.2.2). As we are using the stochastic ranking to embed a preference between the fitness and the constraints, the use of a standard tournament selection procedure where the winner is the individual with the best fitness ignores this preference and thus may not be suitable here. Therefore, we use a modified tournament selection where the winner is the individual with the higher (sorted) rank obtained by stochastic ranking. For empirical verification of this approach, see Section A.4 for a comparison between the two methods.

Environmental selection As discussed above with the parental selection, stochastic ranking is used to embed a preference in the selection of individuals whether they should primarily satisfy the objective or the constraints. Therefore, this preference should also be respected in the environmental selection. For this, stochastic ranking (Section 3.3.2) is used to sort the combined pool of parents and offspring, from which the top $|\mathcal{P}|$ (population size) individuals are taken to the next generation, as opposed to using a purely fitness-based elitism.

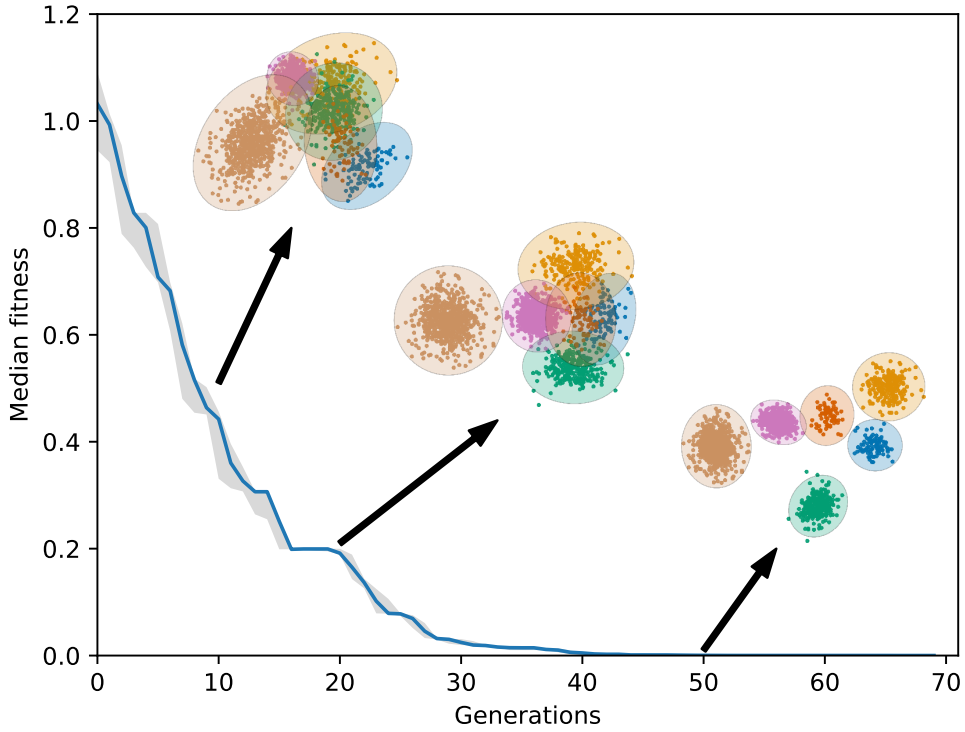


Figure 5.5: Illustration of a single run of HAWKS, and how the cluster structure changes over the generations as the silhouette width is optimized to $s_{\text{target}} = 0.9$. The median fitness (with the first and third quartiles) of the population is shown.

To illustrate the evolution of HAWKS, Figure 5.5 shows the fitness for a single run with $s_{\text{target}} = 0.9$. For this run, any *overlap* was penalized (i.e. $overlap \leq 0$), and a low amount of eccentricity was allowed ($\lambda^{ratio} \leq 10$ was permitted). A population size of 10 was used, with $N = 2000$ and $K = 6$, and run for 70 generations. The best individual (highest fitness, then lowest constraint penalty if tied) is shown at three different generations, with the samples and corresponding normal distribution (to 3 standard deviations) for each cluster is plotted. The first individual is taken from generation 10, where we can see some initial separation (as the initialization is mostly overlapping), represented by a steep decline in fitness. The second individual is taken from generation 20, where we can see a greater degree of separation of the clusters, though there is still some clear overlap. Finally, at generation 50, we can see that the fitness has converged and that the clusters look a lot more separated.

5.2 Evolving controllably difficult clusters

In this section we perform several experiments to evaluate whether HAWKS can generate datasets that result in differential performance across standard, well-known clustering algorithms, and how this diversity compares to existing generators. We illustrate how our generator can be used to produce datasets that consider multiple aspects of problem difficulty.

As our objective function and *overlap* constraint interact in a non-trivial way that varies at different dimensionalities, we first (in Section 5.2.2) investigate this interaction for a deeper understanding of HAWKS and to identify relevant values for the target difficulty at these dimensionalities. Then, in Section 5.2.3 we investigate how controlled variation of the stochastic ranking parameter (P_f) allows us to generate datasets associated with different trade-offs between the silhouette width and *overlap*, and the subsequent differences in dataset difficulty. Finally, in Section 5.2.4 we investigate our choice of the silhouette width as a useful measure to create differential performance among multiple clustering algorithms, and how diversity across those datasets compare to other cluster generators.

5.2.1 Experimental setup

Unless stated otherwise, all experiments use 30 independent runs of the generator, where the best individual (determined by which individual is closest to the target, s_{target}, and in the case of a tie which has the lowest constraint penalty) is selected from each run.

For evaluating the diversity of the datasets based on the clustering performance (in Sections 5.2.3 and 5.2.4), the following algorithms are used:

- K-Means++ (Arthur and Vassilvitskii 2007)
- Single- and average-linkage, run separately with the true number of clusters (K) and $2 \times K$, as these methods can be prone to identifying singleton clusters, thereby potentially performing better with an over-estimation. Generally this should be more important for single-linkage over average-linkage, but we run both for comparative purposes.
- Gaussian mixture models (GMM; [Dempster, Laird, and Rubin 1977])

These algorithms were discussed in further detail in Section 2.4. They are standard algorithms that represent well-understood yet different capabilities for clustering. In general, we can consider K-Means++ and, to a lesser extent, GMM as compactness-based algorithms. Owing to the ability of GMM to find more eccentric clusters and its inherent representation of clusters as Gaussian distributions, without the constraints of HAWKS we expect GMM to perform very well. The linkage-based algorithms represent a slightly different paradigm, with single-linkage very favoured towards the connectedness property of clusters, and the aggregation over data points that average-linkage provides makes it less susceptible to outliers.

To evaluate the clustering performance, the Adjusted Rand Index (ARI) is used (discussed in Section 2.5.1). In brief, the ARI provides a value where 1 (the upper bound) is obtained when every point has been assigned to the same cluster as its label, 0 is the expected value for random cluster label assignment, and negative values indicate a structured misassignment of labels.

5.2.2 Silhouette width at different dimensionalities

The silhouette width utilizes a similarity measure (the Euclidean distance in our case), which as previously discussed (Section 2.1) can be problematic in higher dimensions. We want to identify a range of silhouette width values at different dimensionalities that would be of interest. Specifically, for a desired dimensionality D, we aim to identify which silhouette width values correspond to non-trivial datasets (i.e. not well-separated), where the constraints are hard enough to satisfy that different aspects of cluster quality can exist. As the *overlap* constraint considers only the nearest neighbour, it is less distorted/affected by an increase in dimensionality compared to the silhouette width. For silhouette width values that are close to 0 or 1, the datasets are either pointless or trivial (respectively) to cluster, and thus are not interesting for clustering algorithms and do not create trade-offs with the constraints.

We select two levels of dimensionality: 2D and 20D. For each, we generate two clusters (C_i and C_j) that are hyper-spherical ($\boldsymbol{\Sigma}^{(i)} = \boldsymbol{\Sigma}^{(j)} = \mathcal{I}_D$), with the same mean ($\boldsymbol{\mu}^{(i)} = \boldsymbol{\mu}^{(j)}$) and size ($|C_i| = |C_j|$), but with different samples such that the expected *overlap* ≈ 0.5. These clusters are gradually separated by moving the cluster means in an equal and opposite direction along a single axis/dimension only, as depicted in Figure 5.6.

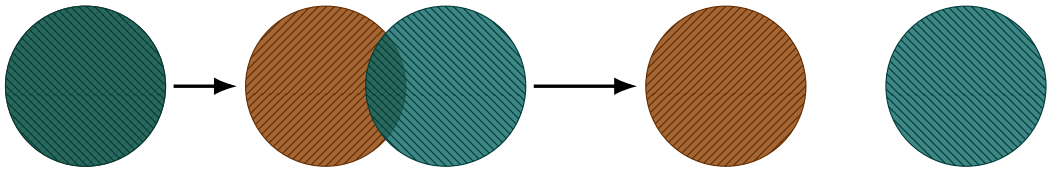


Figure 5.6: Two identical clusters (with different samplings) begin overlapping, and are gradually separated along a single dimension.

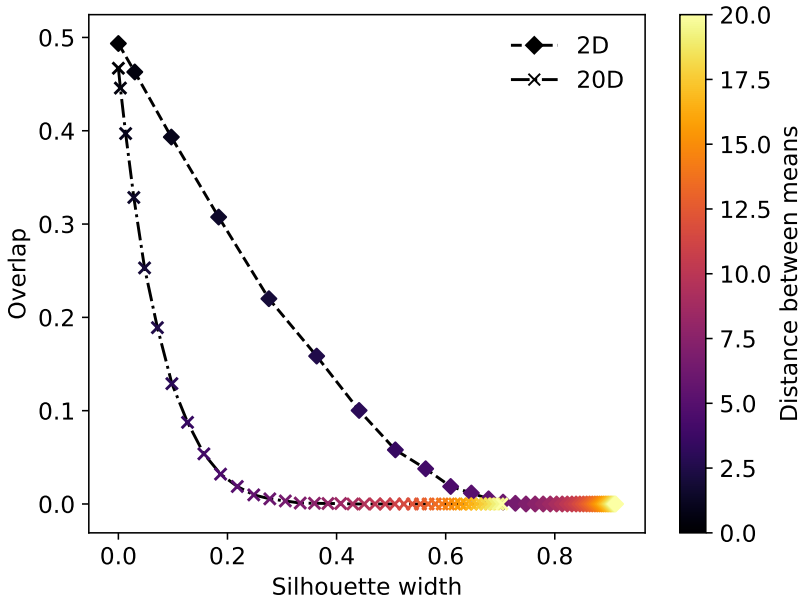


Figure 5.7: Silhouette width vs. *overlap* at 2D and 20D

As expected, Figure 5.7 shows that the silhouette width increases and the *overlap* decreases as the clusters move apart. In this graph, each point represents a step movement apart. In 2D, we see an almost (inverse) linear correlation, where the overlap decreases at a commensurate rate of increase for the silhouette width. In 20D, however, the *overlap* quickly decreases as the clusters are separated, for which there is only a small change in the silhouette width.

As the overlap measures the proportion of nearest neighbours that are assigned to a different cluster, despite the separation in a much smaller proportion of the dimensions this is enough to alter the nearest neighbours such that the overlap measure is affected. The silhouette width measures the actual distance between points, so the separation in one dimension is insignificant until the difference in this dimension dominates the smaller distances in the remaining dimensions, which clearly occurs at a different amount of separation. As such, clusters with no actual overlap (and are thus linearly separable, providing they are convex) are

associated with low silhouette values of $s_{all} \approx 0.35$ in 20D. This is important to consider when selecting values for these parameters that create a trade-off in the optimization, particularly as our mutation operator similarly occurs on a subset of the dimensions (only 1 on average).

5.2.3 The objective-constraint continuum

In Section 5.1.3 we discussed the use of stochastic ranking to trade-off between the objectives and the constraints. Here, we investigate the utility of this mechanism and its ultimate effect on the difficulty of the resulting datasets, as measured by the performance of the four clustering algorithms.

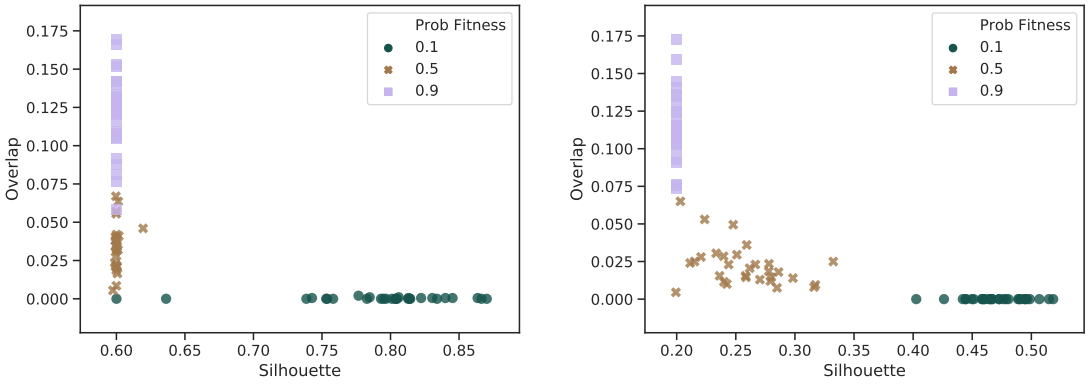
We set the upper bound of our *overlap* constraint as 0, so that it is always penalized. Based on the results in Figure 5.7, we select $s_{\text{target}} = 0.2$ in 20D and $s_{\text{target}} = 0.6$ in 2D. As it is more difficult to generate poorly structured data while avoiding overlap, this should provide conflict between these parameters. We experiment with $P_f \in \{0.1, 0.5, 0.9\}$ to represent a preference towards optimizing the constraints (reducing the violation), no bias, and preferring the objective respectively. The eccentricity constraint is not used here (i.e. $\lambda^{ratio} \geq 1$). The full set of parameters used for HAWKS in this experiment can be found online[5].

In Figure 5.8, we show the values for the silhouette width and the *overlap* at 2D (a), and 20D (b), for the three P_f values. In 2D, we can see that there is no continuous trade-off between the silhouette width and *overlap* for $s_{\text{target}} = 0.6$. We can see, however, that the P_f value did successfully encode a preference in the optimization, as the target is strongly optimized when $P_f = 0.9$ and the *overlap* is greatly minimized when $P_f = 0.1$. Additionally, when using $P_f = 0.5$ a lower overlap was achieved while still successfully optimizing to s_{target} when compared to $P_f = 0.9$, but this did not (for the vast majority of individuals) require an increase in s_{all} to achieve.

In 20D, we can see a broader variety of silhouette width and *overlap* values obtained when $P_f = 0.5$, exhibiting a more direct trade-off. Whereas strong preference for the fitness function or constraints were adhered to, a balanced exploration successfully enabled the discovery of datasets between the silhouette width and *overlap* target values, as there was a clear trade-off between them.

Figure 5.9 shows the performance (ARI) of the aforementioned clustering algorithms at 2D (a) and 20D (b). Here we want to see if the silhouette width

[5] https://github.com/sea-shunned/thesis_material/chp5_stoch-rank.json



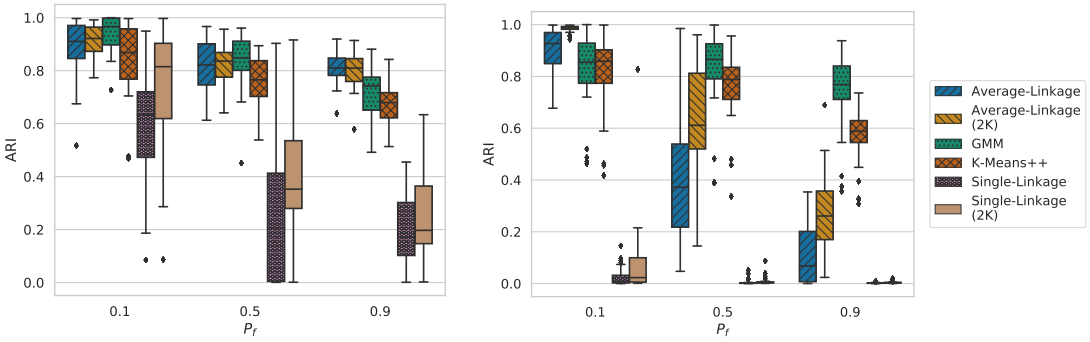
(a) 2D (b) 20D

Figure 5.8: Silhouette width vs. *overlap* for different values of P_f at 2D and 20D. The best individual from each of the 30 runs is shown.

and overlap values above correspond to differences in algorithmic performance, and how this compares across algorithms. For 2D, as we more strictly enforce a lower silhouette width, the performance of all algorithms naturally decreases as the cluster structure becomes less well-separated. Of note is the greater decrease in performance of single-linkage relative to the other algorithms, which is less robust to an increase of overlap between clusters due to its method of identifying clusters to merge based on the minimal distance between any two points in the clusters. The aggregation of average-linkage makes it less susceptible to this, as supported by the higher performance.

In 20D (Figure 5.9b), we can observe generally worse performance of the clustering algorithms, which is likely largely due to the much lower s_{target} and thus overall structure. Single-linkage in particular is mostly unable to cluster the datasets no matter the P_f value, indicating that the lack of structure and close proximity of the clusters at this silhouette width range is particularly challenging. Of interest is the sharp drop in performance of average-linkage as optimizing the fitness is preferred, indicating that (similar to single-linkage) its threshold of required separation between clusters occurs around $s_{all} = 0.3$ in this experiment (shown in Figure 5.8b). The greater robustness and performance of GMM (and, to a lesser degree, K-Means++) is expected as GMM is inherently suited to model our multivariate normal clusters. K-Means++ exhibits a greater drop in performance with increasing *overlap* due to movement of cluster centroids towards denser regions, which may be closer to cluster boundaries due to the overlap.

The P_f parameter is clearly able to greatly modify the optimization process



(a) 2D (b) 20D

Figure 5.9: Performance (ARI) of clustering algorithms for different values of P_f at 2D and 20D.

and fulfill is role in embedding a preference to either the constraints or the objective. This can be particularly useful when trying to generate datasets that more explicitly adhere to constraint settings, or conversely focus on the target. For difficult parameter settings or an *a priori* desire for specific properties, this is a useful lever available in HAWKS.

5.2.4 Benchmark set comparison

Although data generators are useful for creating single or a small number of datasets, the real utility of data generators is in the construction of a benchmark suite of many, diverse datasets. We wanted to preliminarily investigate the diversity of datasets that can be produced by HAWKS using our main proxy for difficulty — the silhouette width. For this, we generated a set of datasets by varying just a few parameters. While it represents a narrow range of the capabilities of HAWKS, it serves to validate whether our choice of the silhouette width as the objective helps generate datasets of varied difficulty. To evaluate the diversity, we compare the generated datasets against two popular synthetic generators (discussed in Section 4.3.2). The parameters for HAWKS (Table 5.1) represent a realistic desire of clusters to have minimal overlap and thus a more "realistic" difficulty, as opposed to a lack of cluster structure.

The key parameters for all three generators are shown in Table 5.1. We show the primary parameters for HAWKS and (where possible) the corresponding values for the other generators. Where parameters cannot be measured in the other generators no value (-) is given. As only the most important subset of

Table 5.1: Generator parameters

Parameter	HAWKS	*HK*	*QJ*
N	2000.2 ± 1.05[i]	1860.25 ± 796.25	599.13 ± 289.21
K	$5, 30$	$4, 10, 20, 40$	$3, 6, 9$
D	$2, 20$	$2, 10, 50, 100$	5–24
P_f	0.5	-	-
Objective target	$0.2, 0.5, 0.8$	0	$0.010, 0.210, 0.342$
λ^{ratio}	≥ 1	-	-
overlap	≤ 0	$0, \leq 0$[ii]	-
Number of datasets	360	160	243

[i] Due to stochasticity in generating random cluster sizes that sum to 2000
[ii] For the 'gaussian' and 'ellipsoidal' generators respectively, as the former does not allow any overlap while the latter penalizes

HAWKS' parameters are shown, the full configuration for this experiment can be found online[6]. With 2 different values each for both K and D, and 3 for s_{target}, there are 12 unique combinations of parameters. Each of these combinations were run 30 times, and the best (defined in Section 5.2.1) dataset was selected from each run to create a combined pool of 360 datasets. We have set $P_f = 0.5$ to create a balanced search between the fitness and constraints, allowing us to set the *overlap* constraint so that it always penalizes, yet retaining flexibility through the stochastic ranking.

We compare against two generators discussed in Section 4.3.2: *HK* (Handl and Knowles 2005b), and *QJ* (Qiu and Joe 2006a). Available online[7], the *HK* generator has two methods of generating datasets, which are referred to as the 'gaussian' and 'ellipsoidal' generators. There are 80 datasets from the 'gaussian' generator with $D \in \{2, 10\}$ and $K \in \{4, 10, 20, 40\}$, and 80 datasets from the 'ellipsoidal' generator with $D \in \{50, 100\}$ and $K \in \{4, 10, 20, 40\}$, covering a range of settings. As discussed in Section 4.3.2, these generators seek to optimize the compactness of the generated clusters and reject/minimize (for 'gaussian' and 'ellipsoidal' respectively) any clusters that overlap (using the same definition as HAWKS).

[6] https://github.com/sea-shunned/thesis_material/chp5_benchmark.json
[7] The 160 datasets can be found at https://personalpages.manchester.ac.uk/staff/Julia.Handl/generators.html

In Qiu and Joe (2006a) the authors define three levels of separation: "close structure", "separated", and "well-separated" (the values for these, which correspond with their measure of separation, are given in Table 5.1). A range of dimensionality is used, from 5 to 24 dimensions, and $K \in \{3, 6, 9\}$. The authors specify differing levels of noisy variables to add further complexity. The authors' R package[8] was used to generate these datasets. Further details of the generators and the parameters they use can be found in Section 4.3.2 and their respective papers.

Results

To analyze the diversity of the datasets, we again look at the clustering performance across the datasets, which is shown for each generator in Figure 5.10. As we are interested in the diversity of performance, we aggregate the ARI values across all datasets for each algorithm. This is not useful for gaining insights into algorithmic performance on specific datasets, but useful for comparing the diversity of the generators. Large variance in the performance for every algorithm would signify an equally diverse set of datasets have been generated, an ideal that would require both a powerful and flexible generator, *and* a systematic generation of an equal number of datasets that exhibit particular properties that are well-understood. For the simple clustering algorithms we use here this is not an unthinkable goal, but clearly the full spectrum of possible cluster structure is such that this ideal may be unreachable.

It is clear that the *QJ* generator has a strong preference for generating compact clusters that are, on the whole, simple for the compactness-based algorithms (GMM and K-Means++) and unsuited for single-linkage. The wide spread of performance for average-linkage highlights its robustness over single-linkage due to its nature of taking an average of the pairwise distances of points between algorithms. The spherical nature of the clusters (and degree of overlap), however, still leads to lower average performance in many datasets, though a wide range is achieved.

On average there is a wider variance in performance for the *HK* generator over HAWKS (through visual inspection of the inter-quartile range), indicating a greater diversity of challenge. To get further insights into the differences between the datasets of these generators, Figure 5.11 shows the silhouette width and

[8] https://cran.r-project.org/web/packages/clusterGeneration/index.html

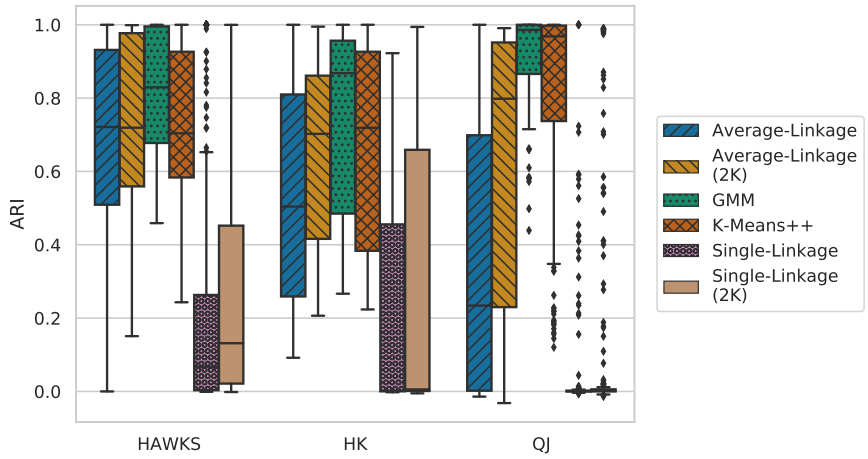


Figure 5.10: Performance (ARI) of clustering algorithms across each of the three generators

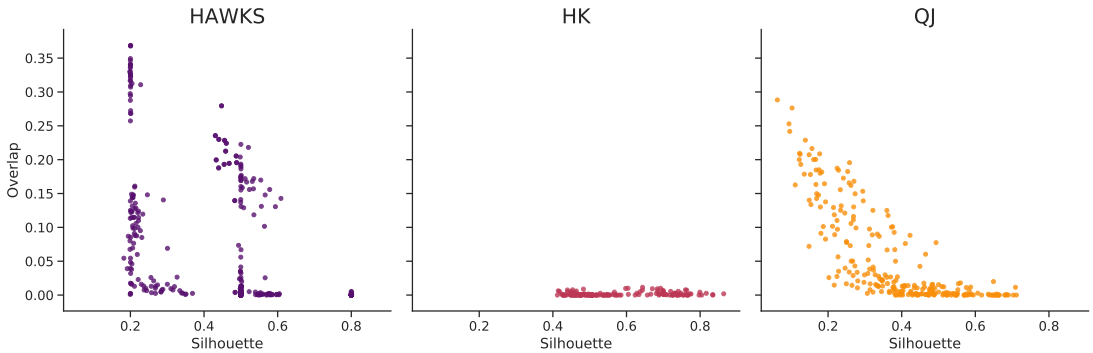
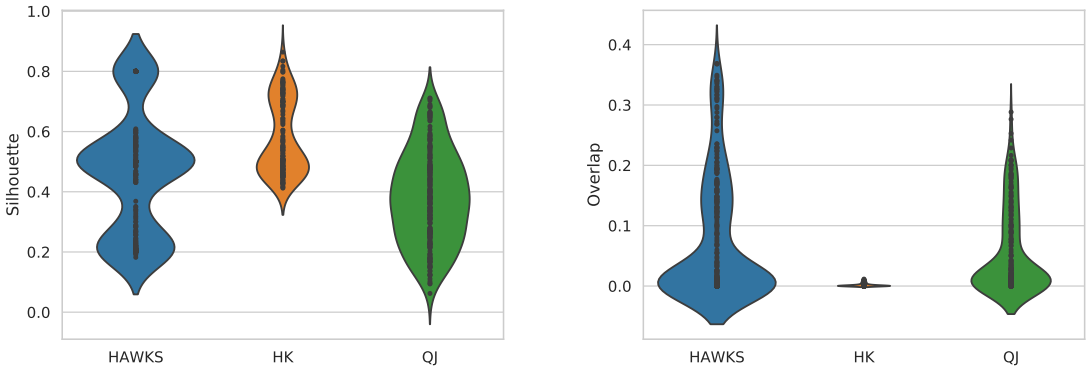


Figure 5.11: The *overlap* and s_{all} calculated for each dataset from the three generators (from left to right: HAWKS, *HK*, and *QJ*).

overlap for each of the datasets from the three generators. The distribution of both the silhouette width and overlap is shown more directly as a violin plot in Figure 5.12.

The *HK* 'gaussian' generator rejects any overlap, and the 'ellipsoidal' generator penalizes overlap in its optimization, which is clearly reflected in the very small deviation from 0 overlap. This may account for the larger spread of performance for (in particular) single-linkage, which is less robust to overlapping clusters. Of interest however, is the higher median performance of all linkage-based algorithms for HAWKS over *HK* (Figure 5.10) despite the lower overlap and higher average silhouette width of the *HK* generator. This may be due to the eccentric clusters that HAWKS can produce (particularly as only half of the generators from *HK* originate from the generator designed for eccentric clusters).



(a) Silhouette width (b) *Overlap*

Figure 5.12: Violin plots showing the distribution of the silhouette width (a) and *overlap* (b) across the datasets from each of three generators. The raw data are shown as markers in each plot.

The banding of HAWKS datasets in Figures 5.11 and 5.12 around the three s_{target} values (0.2, 0.5, and 0.8) and simultaneous existence of datasets between these values show the ability of the generator to balance between the objective function and the constraints. It should be noted, however, that there is a larger diversity of datasets between target values of 0.2 and 0.5 than between 0.5 and 0.8. This may be due to the clear trade-off between silhouette width and *overlap*, where obtaining clusters that overlap at higher silhouette widths becomes much harder than simply optimizing the fitness, decreasing the algorithm's ability to discover datasets in this space. Generating datasets with such properties is easier with smaller clusters (and thus datasets), but as this is a manifestation of a "deceptive" silhouette width (Section 5.1.2) such a dataset is of less interest. The *QJ* generator shows a similar trade-off between silhouette width and overlap, but the 'upper-bound' of overlap that can be generated for a given silhouette width is lower than what is shown by HAWKS. The low silhouette width and high amounts of *overlap* in the *QJ* datasets is the likely cause of the greater spread in performance for K-Means++, and for the poor performance of single-linkage. Explicit parameterization of the overlap with HAWKS may facilitate a wider ability to generate datasets, as the overlap between clusters in the *QJ* generator is only roughly controlled by the level of separation specified in their measure (Qiu and Joe 2006a).

Although Figure 5.10 provided an insight into the *broad* performance of the clustering algorithms for the three generators, the aggregation of this data (and

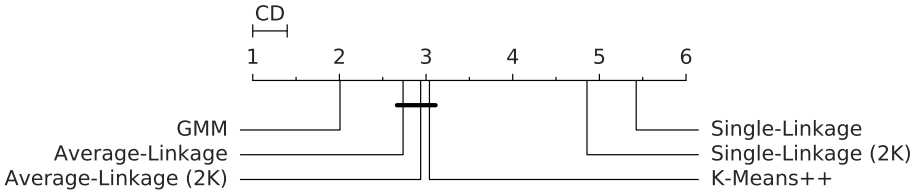
Table 5.2: Generator ARI results

Algorithm	HAWKS	*HK*	*QJ*
Average-Linkage	0.689 ± 0.26	0.536 ± 0.30	0.368 ± 0.38
Average-Linkage ($2K$)	0.741 ± 0.22	0.654 ± 0.24	0.622 ± 0.38
GMM	0.809 ± 0.17	0.749 ± 0.25	0.926 ± 0.10
K-Means++	0.733 ± 0.20	0.647 ± 0.27	0.822 ± 0.24
Single-Linkage	0.220 ± 0.32	0.201 ± 0.31	0.052 ± 0.17
Single-Linkage ($2K$)	0.285 ± 0.34	0.274 ± 0.37	0.091 ± 0.25

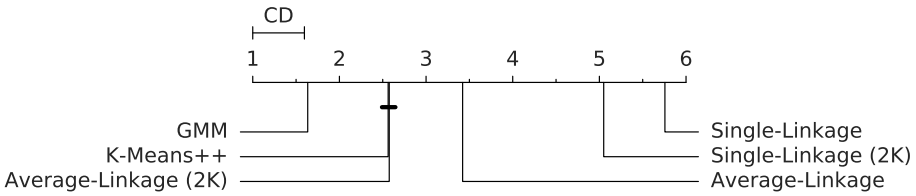
the nature of boxplots) does not give a clear indication of the significance in difference between them. The mean (and standard deviation) ARI values for each algorithm is shown in Table 5.2 to get a different perspective of the variance in performance, though of course this largely corroborates the trend seen in Figure 5.10.

To ascertain the significance of the differences between them, we follow the procedure outlined in Demšar (2006) for comparing multiple algorithms across multiple datasets. In short, we perform a Friedman test that ranks each algorithm for each dataset, where the null hypothesis is that all algorithms are equal and therefore have equal ranks. Rejection of this null hypothesis indicates that at least one algorithm is significantly different, and to identify which we need a post-hoc test. The two-tailed Nemenyi test (Nemenyi 1962) is used for this, which calculates the critical difference (CD) — the CD is the minimum that two average ranks must differ by to be significantly different (unless stated otherwise, we use $p = 0.05$ significance level for the null hypothesis). Demšar (2006) presents the CD diagram as a method to illustrate the average ranks of the methods, highlighting any significant differences. In these diagrams, a solid line that connects two or more algorithms indicates that those methods are not significantly different from each other (i.e. their average ranks do not differ by more than the calculated CD).

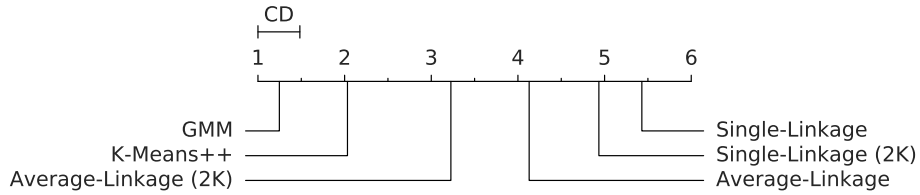
It is important to note here that Benavoli, Corani, and Mangili (2016), among others, argue that the use of mean-ranks post-hoc tests should be discouraged, as the CD is computed using all algorithms and thus the statistical significance between two algorithms actually depends on the performance of the *other* algorithms. For our scenario, this argument would hold if we introduced multiple



(a) HAWKS



(b) *HK*



(c) *QJ*

Figure 5.13: Critical difference (CD) diagrams for each of the three generators, showing the average rank for each algorithm across the datasets from that generator. Solid lines connect algorithms which are not significantly different from each other according to a two-tailed Nemenyi test.

algorithms that we believed would do either well or poorly across all datasets, thus inflating or deflating (respectively) the ranks of the remaining algorithms. As we are evaluating the diversity of difficulty across these datasets, which is unknown, such issues are not relevant here.

In Figure 5.13 we show CD diagrams for each of the generators to provide a clearer view of performance across the algorithms. Figure 5.13a shows that for HAWKS, GMM is significantly the best algorithm, but that the two average-linkage variants and K-Means++ are tied for 2$^{\text{nd}}$ place. Owing to the Gaussian representation used in HAWKS, it is unsurprising that GMM performs the best. With K-Means++ and average-linkage performing equally well, it signifies that

there is sufficient variance in the amount of overlap and eccentricity to present challenges for these two algorithms. Single-linkage is the worst performer for all three generators, which was also clear from the boxplots (Figure 5.10).

The rankings for the *HK* datasets are shown in Figure 5.13b, where we can also see the superiority of GMM. Giving double the true K value has lead to a significantly better performance for average-linkage (in contrast to the datasets produced by HAWKS), which is likely due to a smaller degree of eccentricity in the 'gaussian' datasets (as spherical clusters are more challenging for the linkage-based methods). Finally, Figure 5.13c shows the rankings for the *QJ* generator. Here, the algorithms show a clear ranking with GMM nearly achieving a perfect average rank, closely followed by K-Means++, highlighting the stronger bias towards these compactness-based algorithms. The performance of the remaining linkage algorithms follow a similar pattern with the $2K$ variants performing better, and single-linkage overall performing the worst.

The tighter rankings (and increased central tendency to the average rank of 3.5) of the algorithms on datasets generated by HAWKS highlights a more even spread of performance across the algorithms, with the well-ordered rankings for the *QJ* datasets clearly demonstrating the lack of variance (in terms of challenges for different algorithms) among them. The *HK* datasets show a ranking distribution in between the other two, which is consistent with the previous results (Table 5.2 and Figure 5.10).

Instance space

To visualize the datasets according to their problem features, we construct a simple instance space using a few features: the dimensionality (D), *overlap*, and silhouette width (s_{all}) of the dataset. Although the work of Smith-Miles uses many more features in the construction of an instance space (Smith-Miles and Bowly 2015; Smith-Miles et al. 2014; Smith-Miles and Lopes 2012), we use the three aforementioned features as they can be meaningfully and easily measured across the datasets from the 3 generators. To easily visualize this space, we use principal component analysis (PCA) to project the values of the three features down to 2D.

Figure 5.14 shows the instance space, with the datasets from each of the generators highlighted. Visually, both HAWKS and QJ cover a large amount of the space, indicating greater diversity (across the problem features) for these

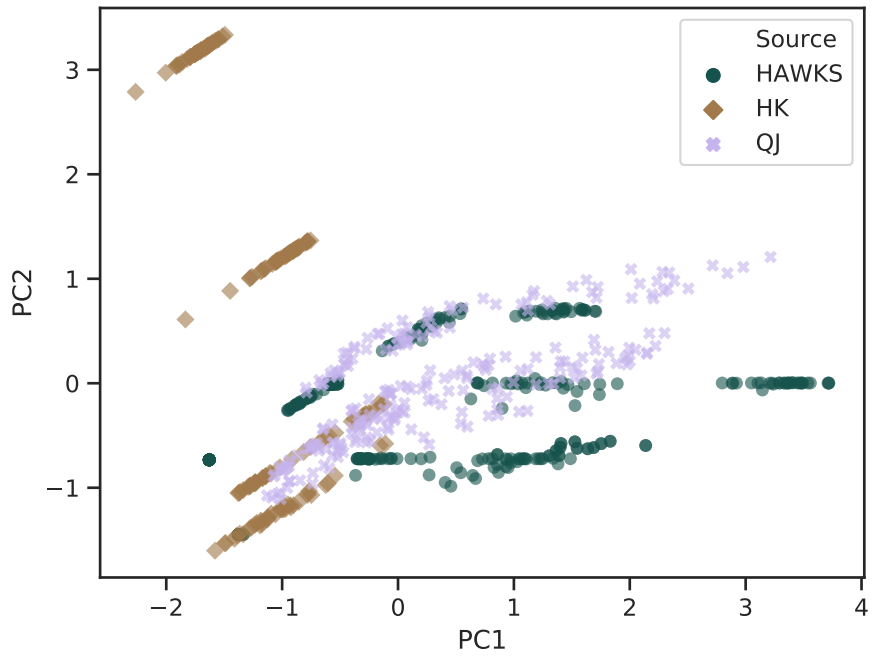
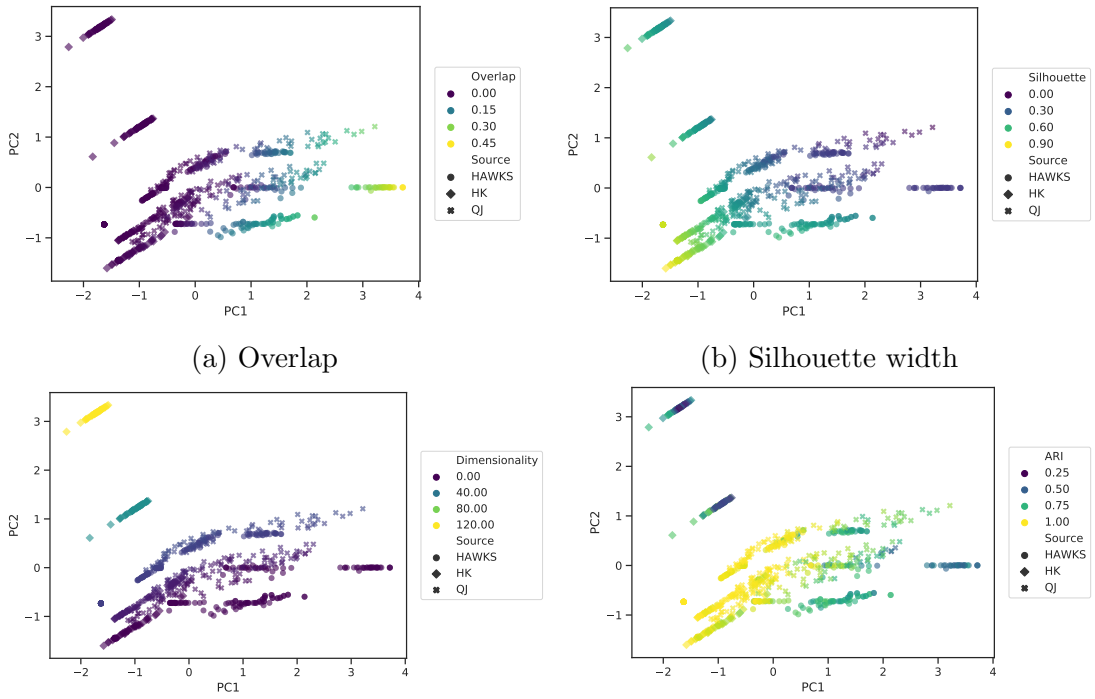


Figure 5.14: The datasets from each generator ('source') plotted on an instance space. This space is constructed using the silhouette width, *overlap*, and D for each dataset and reduced to 2D using PCA.

generators. The majority of the variance in the 2$^{\text{nd}}$ principal component is due to the dimensionality of the data, where the HK generator extends significantly higher. In addition, the stronger banding of the HK generator indicates more similarity between the various instances produced using different settings (of K and D). Of particular note is the groups of datasets formed by HAWKS (contrast to the more contiguous grouping of QJ), indicating that the different parameter combinations have a distinct effect.

The instance space for different properties is shown in Figure 5.15. In Figures 5.15a to 5.15c, we show the features that were used to construct the instance space so we can see how they vary across the space. Figure 5.15d shows the best ARI found by any algorithm for each dataset. Comparing the silhouette width (b) and ARI (d) values across the datasets, we can see a loose correlation between the two (as expected). Of note are the datasets produced by the 'ellipsoidal' (HK) generator (the two bands at a higher dimensionality), which proves to be significantly harder than the datasets from its 'gaussian' counterpart. As these datasets are not associated with a particularly different overlap or silhouette width, it is likely that the dimensionality (a known problem in clustering)



(a) Overlap

(b) Silhouette width

(c) Dimensionality

(d) ARI

Figure 5.15: The constructed instance space, with values for different properties displayed. The first three figures — (a), (b), and (c) — are the features used by PCA to construct this space. The best ARI found for each dataset is shown in (d).

may be the cause of this difference, though it may also be due to other problem features we have not captured. One such feature we have not captured is the eccentricity, particularly as the *HK* 'ellipsoidal' generator was explicitly designed for that, and there is a wide spread of performance in these datasets that is not reflected in the instance space.

As also shown in Figures 5.10 and 5.12, the silhouette width does not provide a complete indication of difficulty. Nevertheless, it seems a useful lever in configuring the difficulty of datasets produced by HAWKS.

5.3 Summary

This chapter has described an evolutionary algorithm to generate synthetic data to facilitate more specific empirical comparisons, thus obtaining further insights into clustering algorithms. A simple, single-objective version of this generator (named HAWKS) was shown to be flexible, and offer more parameterization and thus control over the difficulty and properties of the resulting datasets when compared to existing generators. It is clear, however, that some improvements are needed for a more capable generator:

1. We previously discussed the need for the initialization scheme to generate overlapping clusters as (particularly in higher dimensions) the mutation operator has a bias towards increasing the silhouette width, and will have difficulty bringing high-dimensional clusters closer together. A more capable operator will assist with generating datasets at higher dimensionalities.

2. We have not explored the range of datasets that HAWKS can produce when varying its other parameters, in particular the λ^{ratio} constraint can be used to bias the generation of clusters that are either more compact (when lower) or eccentric (when higher), and the effect that this could have to clustering algorithms.

3. The instance space used a limited number of features in its construction — can we identify other features that can provide a deeper insight into the diversity and different properties of these problems?

4. After these modifications, can we (through more explicit and varied parameterization) generate a more comprehensive set of datasets that could

be used for algorithmic insights? Is this diversity reflected in the instance space and/or algorithmic performance?

In the next chapter, we address these issues and further develop HAWKS.

Chapter 6

Evolving Difficult Synthetic Clusters II

In Chapter 5, we introduced HAWKS, a synthetic data generator that optimizes the datasets according to an objective (the silhouette width) and some constraints (the percentage of points overlapping and the maximal eccentricity of a cluster), allowing for the parameterization of cluster properties that partially represent the difficulty of the dataset. We identified some shortcomings and areas for future expansion that we explore in this chapter, mainly: addressing the bias of the mutation operator that scales with dimensionality (Section 6.1); a fuller set of problem features and generation of more varied datasets (Section 6.2); and, generating datasets for specific algorithms (Section 6.3).

6.1 Mutating clusters in higher dimensions

In Section 5.1.3, we noted that the current mutation operator for the cluster means ($\boldsymbol{\mu}$) is decreasingly useful as the dimensionality increases due to stochastic movement of the mean, resulting in a bias towards increasing the silhouette width (as the clusters drift apart). In this section, we implement and compare several different mutation methods that try to explicitly embed directionality into the random movement to avoid this bias.

6.1.1 Candidate mutation operators

To reduce the bias towards clusters drifting apart in higher dimensions, the addition of explicit directionality to the operator can guide whether the movement of clusters is either away from or towards existing clusters, increasing and decreasing the silhouette width respectively. As such, a new mutation operator would need to incorporate the position of at least one other cluster in the random perturbation. For this, several new operators have been proposed, which are described in the following sections, accompanied by an illustration of how they work.

In each illustration we show a hypothetical dataset of $K = 5$ clusters, with only the means of each cluster shown. The cluster means are represented by different symbols: $\boldsymbol{\mu}^{(i)}$ signifies the mean that is being mutated; $\boldsymbol{\mu}^{(n)}$ signifies the mean(s) used to create directionality in the mutation; $\bar{\boldsymbol{\mu}}$ is the global centroid (of all data points); and, $\boldsymbol{\mu}$ is a placeholder for the means of the other clusters in the dataset that are not involved in the mutation, which are unlabelled for simplicity. In each illustration, a shaded area (or line) indicates the space in which the mean can mutate within and the criteria (where applicable) to select that area.

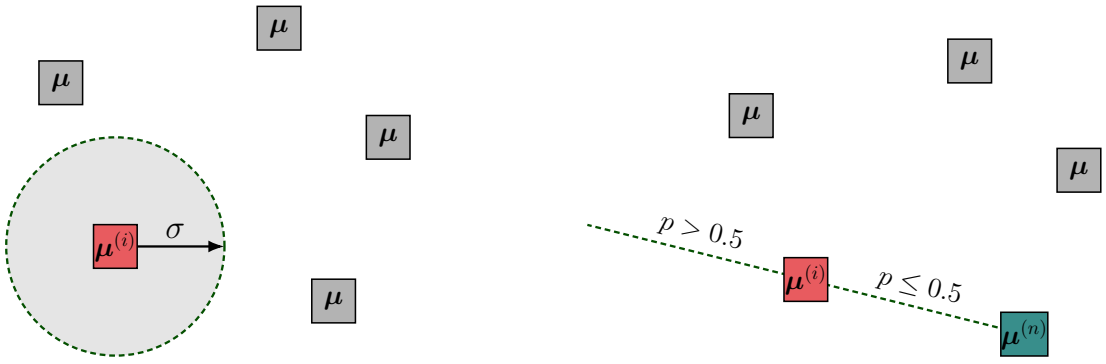
Original operator

In Equation 5.7 we defined the original mean mutation operator for the ith cluster as $\boldsymbol{\mu}'^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}^{(i)}, \sigma^2)$, where $\boldsymbol{\mu}'^{(i)}$ is the new mean, $\boldsymbol{\mu}^{(i)}$ is the current mean, and σ^2 is the variance (width) of the multivariate normal distribution. In other words, the new mean is simply sampled from a normal distribution around the current mean. By default, each dimension is sampled separately from a univariate distribution and has a $\frac{1}{D}$ chance of occurring (to avoid inflated mutation rates). The illustration in Figure 6.1a shows the total possible area that the mean can mutate to.

"Rails" operator

As a simple baseline for the inclusion of directionality, a single cluster is selected and the current cluster moves either away from or towards that cluster (without any lateral movement, i.e. on a rail) with a random weighting, w_1.

To select a random cluster that is different from the current one, let n be a random integer from the set $\{1, \ldots, K\} \setminus \{i\}$. The mean of the current cluster,



(a) Original (b) "Rails"

Figure 6.1: Reachable area for mutating $\boldsymbol{\mu}^{(i)}$ using the original and "rails" mutation operators.

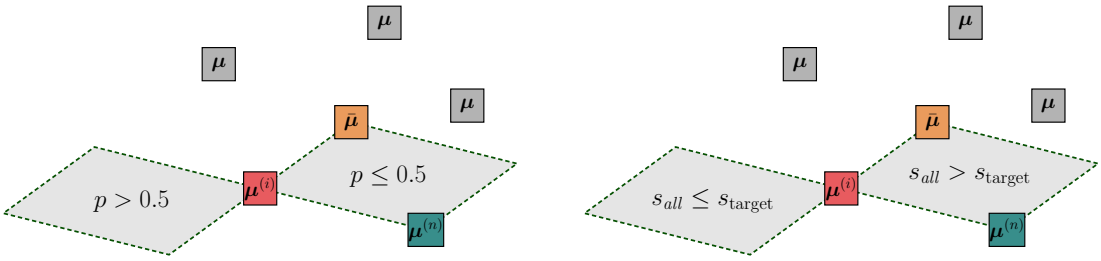
$\boldsymbol{\mu}^{(i)}$, can be mutated as follows:

$$\boldsymbol{\mu}'^{(i)} = \begin{cases} \boldsymbol{\mu}^{(i)} + w_1(\boldsymbol{\mu}^{(n)} - \boldsymbol{\mu}^{(i)}) & \text{if } p \leq 0.5 \\ \boldsymbol{\mu}^{(i)} - w_1(\boldsymbol{\mu}^{(n)} - \boldsymbol{\mu}^{(i)}) & \text{if } p > 0.5 \end{cases} \tag{6.1}$$

where $\boldsymbol{\mu}^{(n)}$ is the mean of the random cluster n, $w_1 \sim \mathcal{U}(0,1)$ is a random weight, $\boldsymbol{\mu}^{(n)} - \boldsymbol{\mu}^{(i)}$ is the difference between the cluster means, and $p \sim \mathcal{U}(0,1)$ is a random uniform probability. Figure 6.1b shows the continuous line upon which the mean can mutate along, the direction of which is determined by a coin-flip (p), and the position along that line is determined by w_1.

PSO-inspired mutation with random directionality

For a mutation that covers more of the space, we take inspiration from another EA paradigm: particle swarm optimization (PSO). In the original PSO mutation, each particle is updated by incorporating their current position, the best position that particle has ever had, and the best position ever found by any particle. These latter two best positions are weighted by random coefficients in order to create a random movement of the particles. For further details, see Kennedy and Eberhart (1995) and Kennedy (2010). We use this as inspiration by viewing the clusters as particles, and try to mutate the location of the cluster based on the position of another cluster and some global representative. As our fitness is derived from a combination of all particles, the original notions of personal and global best are not applicable here, but the concept of independently weighting



(a) Random directionality (b) Informed directionality

Figure 6.2: Reachable area for mutating $\boldsymbol{\mu}^{(i)}$ using the PSO-inspired mutation operators

both a single point and an aggregated one is.

By updating the mean using a randomly weighted combination of an existing cluster and the global mean ($\bar{\boldsymbol{\mu}}$) of all data points, we can create a random movement of the cluster that still takes into account the position of the existing clusters. By incorporating the global mean, we can avoid generating well-separated groups of clusters that can deceive the silhouette width (Section 5.1.2). To ensure that the location of the global mean is calculated in a way meaningful to the fitness, we calculate the mean across all *data points* (not across cluster means), as it is the former that is used in the calculation of the silhouette width (and thus fitness). As a result, relative cluster sizes are incorporated into the mutation.

The mean of the current cluster, $\boldsymbol{\mu}^{(i)}$, can be mutated as follows:

$$\boldsymbol{\mu}'^{(i)} = \begin{cases} \boldsymbol{\mu}^{(i)} + [w_1(\boldsymbol{\mu}^{(n)} - \boldsymbol{\mu}^{(i)}) + w_2(\bar{\boldsymbol{\mu}} - \boldsymbol{\mu}^{(i)})] & \text{if } p \leq 0.5 \\ \boldsymbol{\mu}^{(i)} - [w_1(\boldsymbol{\mu}^{(n)} - \boldsymbol{\mu}^{(i)}) + w_2(\bar{\boldsymbol{\mu}} - \boldsymbol{\mu}^{(i)})] & \text{if } p > 0.5 \end{cases} \tag{6.2}$$

where $w_2 \sim \mathcal{U}(0, 1)$ is the second random weight. The area that the mean could mutate to (depending on the coin-flip) is illustrated in Figure 6.2a. This operator will be referred to as PSO-random.

PSO-inspired mutation with informed directionality

By using a coin-flip to determine whether the cluster should move towards or away from existing clusters, there arises obvious scenarios where the existing clusters may be very far apart and a closer structure is desired. An unfavourable coin-flip moves this cluster further away, wasting a perturbation and thus fitness evaluation. By using whether the current silhouette width (s_{all}) of the individual is above or below the target (s_{target}), we can move the cluster centre in the direction

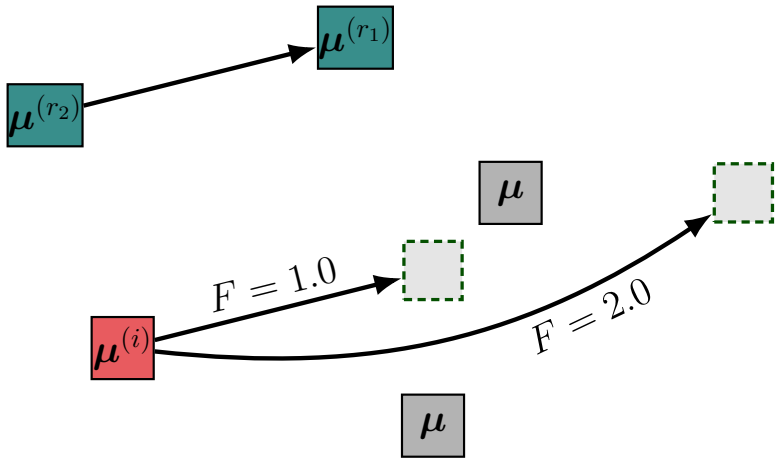


Figure 6.3: Reachable area for mutating $\boldsymbol{\mu}^{(i)}$ using the DE-inspired mutation operator. Note that, unlike the previous operators, as a constant scaling factor (F) is used, only a single point can be reached and not a variable area.

that will have the best chance of improving the fitness.

The mean of the current cluster, $\boldsymbol{\mu}^{(i)}$, can be mutated as follows:

$$\boldsymbol{\mu}'^{(i)} = \begin{cases} \boldsymbol{\mu}^{(i)} + [w_1(\boldsymbol{\mu}^{(n)} - \boldsymbol{\mu}^{(i)}) + w_2(\bar{\boldsymbol{\mu}} - \boldsymbol{\mu}^{(i)})] & \text{if } s_{\text{all}} > s_{\text{target}} \\ \boldsymbol{\mu}^{(i)} - [w_1(\boldsymbol{\mu}^{(n)} - \boldsymbol{\mu}^{(i)}) + w_2(\bar{\boldsymbol{\mu}} - \boldsymbol{\mu}^{(i)})] & \text{if } s_{\text{all}} \leq s_{\text{target}} \end{cases} \tag{6.3}$$

This is illustrated in Figure 6.2b, where we can see that if the individual's silhouette width is higher than the target, the mean is more likely to mutate towards the other clusters. This operator will be referred to as PSO-informed.

DE-inspired mutation

Taking inspiration from yet another EA paradigm, differential evolution (DE), we view the existing clusters as individual vectors which can help in the creation of a "donor vector" (the new mean) from a "target vector" (the current mean). The classical DE mutation operator combines three existing individuals in order to create a new individual (Fleetwood 2004; Ortiz and Xiong 2014). We adapt this to generate a new cluster mean from existing ones as follows:

$$\boldsymbol{\mu}'^{(i)} = \boldsymbol{\mu}^{(i)} + F(\boldsymbol{\mu}^{(r_1)} - \boldsymbol{\mu}^{(r_2)}) \tag{6.4}$$

where i, r_1, and r_2 are distinct indices of a cluster, and F is a constant factor in the range $[0, 2]$. This operator is illustrated in Figure 6.3, where we can see

distinct points that the mean can mutate to. Unlike with the previous operators, the randomness occurs only in terms of which other means are selected, such that the movement vector for the current mean is a fixed multiple of the vector between the randomly selected means. This is illustrated by, for the two means randomly selected, the two new means corresponding to $F = 1.0$ and $F = 2.0$. As a result, the number of possible locations that a cluster can mutate to is finite and related to the number of clusters.

6.1.2 Experimental setup

The main issues that the new mutation operator needs to address are: convergence in higher dimensions; and, the ability to converge regardless of the initialization. Therefore, we construct four scenarios where the initialization generates clusters that are either overlapping or apart, and we set a target silhouette width (s_{target}) as either low (0.2) or high (0.9). Thus, we test not only the ability of each operator to move clusters together or apart at different dimensionalities, but also whether they are able to maintain such states over many generations (testing their stability). Each of the five operators listed in Section 6.1.1 will be run 30 times in each of these four scenarios using $D \in \{2, 50\}$ dimensions.

The parameter ranges for the different operators are set as given in the equations of the previous section. For the DE-inspired operator, we set $F = 1$ to encourage movement of the clusters without changes that are too large. The full configuration for this experiment can be found online[1].

We expect that all operators should provide some improvement over the original operator with regards to optimizing the fitness in higher dimensions, with one possible exception. The DE-inspired operator moves clusters according to the vector between any two random clusters, which in higher dimensions provides many potentially unhelpful directions (in terms of increasing or decreasing the fitness). This may harm the operator's capabilities in higher dimensions, but it is unclear whether this is to the same extent as the original operator. The "rails" operator should be able to move clusters both towards and away from each other, but the simplicity (not taking into account overall structure) may lead to the aforementioned (in Figure 5.2) issues of a misleading silhouette width, where the average obscures subsets of clusters that are close together. The inclusion of

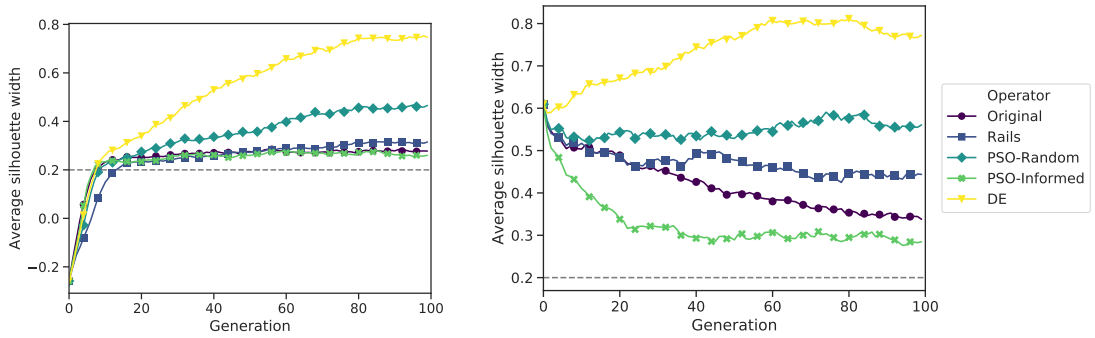
[1] https://github.com/sea-shunned/thesis_material/chp6_mut.json

direct fitness information into the PSO-informed operator should increase convergence speed, though this may come at the cost of increased violation of the *overlap* constraint.

6.1.3 Results

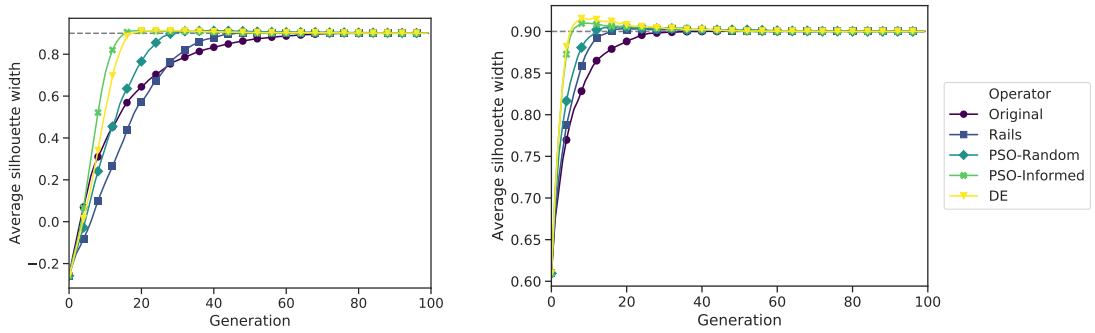
Figure 6.4 shows convergence plots (the average silhouette width, s_{all}, for each generation across the 30 runs) for the four different scenarios in 2D. The dashed line on each plot shows the target silhouette width (s_{target}). The scenario with overlapping clusters and a low silhouette width target ($s_{\text{target}} = 0.2$) is shown in Figure 6.4a. The overlapping initialization is such that the initial silhouette width is negative, requiring the mutation operators to move the clusters apart. All mutation operators are quickly able to do this, rapidly increasing the silhouette width, but we see a difference in the stability after this as some operators (predominantly DE and PSO-random) further increase the silhouette width above s_{target}, highlighting a mechanistic bias. In Figure 6.5a we can see the minimization of the *overlap* constraint, which is naturally greater for the operators that increased the silhouette width. This indicates a difference in the relative ease of satisfying s_{target} and minimizing *overlap* between the operators, particularly with the lack of pressure (afforded by $P_f = 0.5$) towards either one. With such a low s_{target}, there is a strong inverse correlation between the *overlap* and fitness. The PSO-informed operator has an explicit drive to avoid a drift of the silhouette width away from the target, but the vast difference in drift for the DE-inspired operator illustrates a clear behavioural difference.

In Figure 6.4b, we have the same low s_{target} but with the initialization scheme that samples the initial cluster centres further apart (thus with an initial average silhouette width far above s_{target}). As this scenario explicitly tests the ability of the operators to bring clusters *together*, we can see a stark difference between them (as hinted previously). The DE-inspired operator is unable to decrease the fitness at all, and just minimizes the *overlap* (shown in Figure 6.5b). As the DE operator can only move by a fixed multiple of the vector between two of the current clusters, there is a lack of nuance when moving clusters (which is important when making smaller improvements to a fit individual). The PSO-random operator is unable to converge to s_{target}, but as the PSO-informed operator is able to rapidly converge we can conclude that this is due to a lack of directionality towards directly improving the fitness as opposed to an inability of the mechanism



(a) Overlapping, $s_{\text{target}} = 0.2$

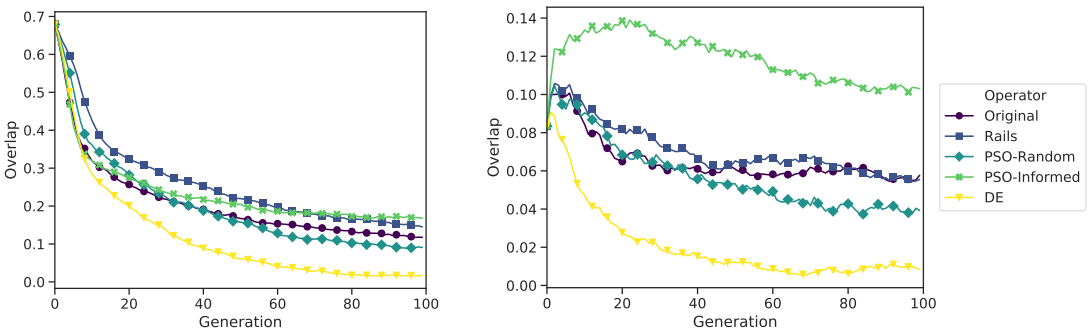
(b) Apart, $s_{\text{target}} = 0.2$



(c) Overlapping, $s_{\text{target}} = 0.9$

(d) Apart, $s_{\text{target}} = 0.9$

Figure 6.4: Convergence plots showing the average fitness across the 30 runs for each generation for each of the four scenarios (initializing the clusters together and apart, both at a low and high s_{target}) in 2D. The dashed line shows s_{target} in each scenario.



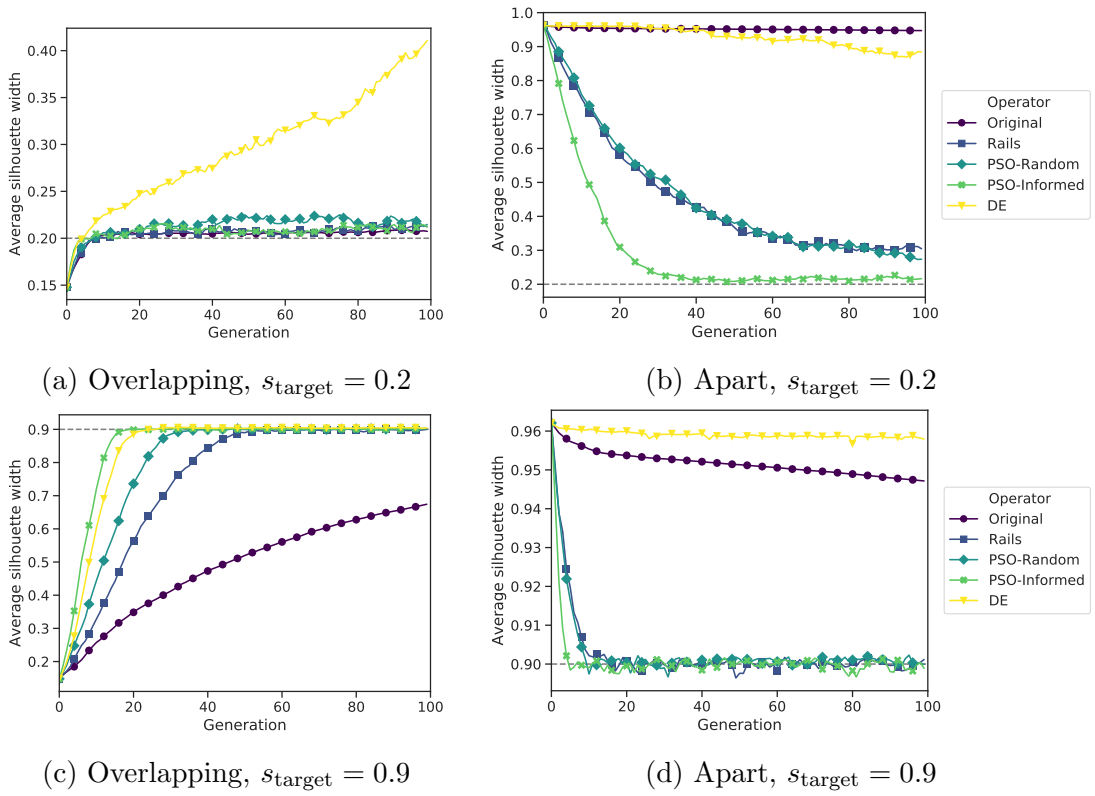
(a) Overlapping, $s_{\text{target}} = 0.2$ (b) Apart, $s_{\text{target}} = 0.2$

Figure 6.5: Convergence plots showing the *overlap* constraint for two of the scenarios in 2D.

in moving the cluster means. The slow convergence of the original operator highlights its mechanism of taking small random steps, an issue we know grows with the dimensionality.

There is no significant difference in the speed of convergence between the operators when optimizing for $s_{\text{target}} = 0.9$, independent of the initialization used (Figures 6.4c and 6.4d). As expected due to its explicit embedding of directionality according to the fitness, the PSO-informed operator converges the quickest, and our original operator the slowest (though the difference here is minimal). None of the operators drift once converged, as this target also satisfies the *overlap* constraint, meaning that there is no further selection pressure away from s_{target}.

Figure 6.6 shows the convergence plots for the same four scenarios in 50D. Looking at Figure 6.6a, we again have the scenario where we initialize with overlapping clusters and a low s_{target}. Owing to the nature of silhouette width in higher dimensions (which we explored in Section 5.2.2), the actual silhouette width of the initial population is closer to (but still below) s_{target}. All operators (apart from the DE-inspired operator) are able to maintain a silhouette width close to the target, which is at least better than the initialization. The real difficulty is seen in Figure 6.6b, where the initialization results in a silhouette width of close to 1, requiring the clusters to be brought together. Our original operator is unable to make any improvement at all in this higher-dimensional space, with the DE-inspired operator barely able to do better. As the DE operator uses the direction and magnitude of the vector between two random clusters, it does not necessarily move the cluster being mutated in the direction of other clusters



(a) Overlapping, $s_{\text{target}} = 0.2$

(b) Apart, $s_{\text{target}} = 0.2$

(c) Overlapping, $s_{\text{target}} = 0.9$

(d) Apart, $s_{\text{target}} = 0.9$

Figure 6.6: Convergence plots showing the average fitness across the 30 runs for each generation for each of the four scenarios in 50D. The dashed line shows s_{target} in each scenario.

(particularly in higher dimensions), thus confirming our earlier concern. The remaining operators are able to significantly decrease the fitness (as they do move the cluster with respect to the other clusters), and the explicit directionality utilized by the PSO-informed operator allows rapid convergence to the target. The small increases of the silhouette width in the later generations correspond with larger minimizations of the *overlap* (not shown).

The ability of our original operator to move clusters away from each other in higher dimensions is highlighted in Figure 6.6c, which is clearly contrasted with the proposed operators that are all able to converge significantly faster. Once again, utilizing the individual's current silhouette width allows the PSO-informed operator to consistently move the clusters apart, converging rapidly. The speed of convergence for the DE-inspired operator is likely due to its stepsize being based on the distance between two random clusters, thus enabling it to rapidly move clusters apart. When the clusters are initialized further apart (Figure 6.6d), similar behaviour is seen for the high s_{target} scenarios as with the low s_{target}. The

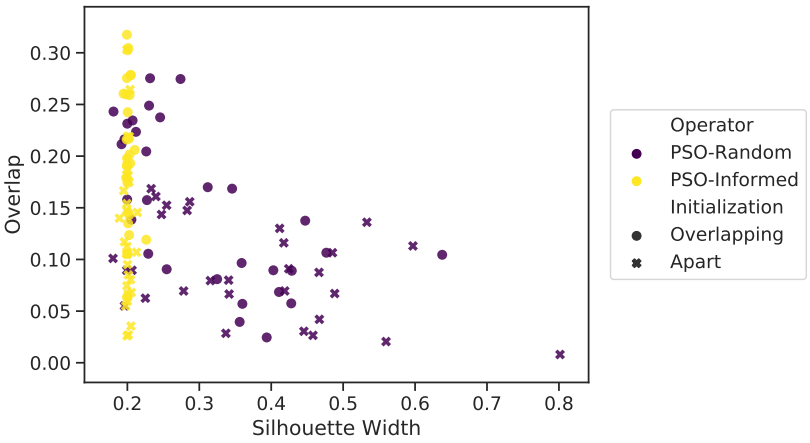


Figure 6.7: The silhouette width against *overlap* for the two PSO-inspired operators, showing the bias towards the fitness function that the PSO-informed operator inherently provides. The best individuals from each of the 30 runs are shown.

initial silhouette width is very close to the target, yet our original and the DE-inspired operators are still unable to significantly improve the fitness as they begin *above* the target. It is likely that in this scenario, the stepsize (movement of the cluster mean) is too large for the DE-operator to be useful, and too small and undirected for our original operator.

6.1.4 Experimental summary

All of the proposed operators confer some advantage over the original operator in higher dimensions. The DE-inspired operator, however, struggles to bring clusters closer together, a deficiency shared with the original operator, due to a lack of directionality and inappropriate stepsize. The rails operator appears to offer no advantage over either of the PSO-inspired operators, of which PSO-informed exhibits faster convergence at the cost of embedding a preference to the objective over the constraints (as illustrated by the higher *overlap* values obtained). Focussing on the two PSO-inspired operators, in Figure 6.7 we can see a difference in the spread of datasets between the silhouette width and *overlap*, thus depending on the intended properties of the datasets one operator may be preferable over the other. The bias towards satisfying the objective with the PSO-informed operator also creates a conflict with P_f, exacerbating the difficulty of parameter setting, and thus may be less useful in general.

6.2 Expanding the instance space

At the end of Chapter 5, we highlighted that our experiments had varied a subset of the parameters available in HAWKS. One issue in testing a wider set of parameters is identifying which are important in terms of generating varied, interesting, and challenging datasets. On top of the usual parameter tuning (Section 3.4.1) issues with EAs for the population size, mutation rates etc., with HAWKS we have the additional layer of tunable parameters and constraints specific to clustering. A deeper understanding of the relative importance of these parameters is needed to know which levers we need to pull, and how hard we need to pull them when generating a more comprehensive benchmark set.

In this section, we build upon the experiment in Section 5.2.4 by identifying additional problem features and other datasets to compare against, using a wider range of parameters within HAWKS. With this instance space, we then attempt to directly target an area in this space to generate datasets with properties that HAWKS had not previously produced, assessing the flexibility and parameterization of HAWKS.

6.2.1 Expanding the problem features

A notable issue with the instance space presented in Section 5.2.4 was both the low number of features used in its construction and that these features could be directly modified in HAWKS. As noted in Section 4.2, the algorithm selection problem frames the problem features as the set of measurable properties of the instances (datasets) that capture their difficulty. We have previously discussed (in Section 4.3.4) the difficulty, or perhaps impossibility, of such a complete set being designed for clustering. Nevertheless, the identification and subsequent inclusion of more problem features should improve the utility of the instance space, leading to a more robust visualization of the datasets and their relative differences.

The inclusion of more cluster validity indices (Section 2.5.1) into the feature set is one obvious way of providing more direct information about the clustering-related properties of the datasets (as opposed to more general data statistics). An issue with using these indices as features, however, is in our projection in the instance (particularly as we use PCA). Cluster validity indices can be correlated for many situations, yet for certain cluster structures the subtle differences between them may result in uncorrelated variation (Arbelaitz et al. 2013). PCA

may overexaggerate the contribution of these features and thus skew the resulting instance space, so some care needs to be taken beforehand to ensure that we do not select multiple, highly correlated indices.

In the following paragraphs we describe the additional problem features that have been added to the dimensionality and average silhouette width, which were used in Section 5.2.4 (the *overlap* has been replaced as a problem feature) to create a set of 7 features.

Connectivity Handl and Knowles (2007) proposed the connectivity measure (which was a modified version of the measure introduced in Ding and He [2004]) to evaluate the extent to which data points have been assigned to the same cluster as their nearest neighbours. Denoting L as the neighbourhood size (i.e. the number of nearest neighbours that we consider), the connectivity is calculated as follows:

$$\frac{1}{N}\sum_{\mathbf{x}\in X}\sum_{j=2}^{L}\rho(\mathbf{x}, n_{\mathbf{x}}(j)), \tag{6.5}$$

where $n_{\mathbf{x}}(j)$ is data point $\mathbf{x}$'s jth nearest neighbour, and:

$$\rho(\mathbf{x}, n_{\mathbf{x}}(j)) = \begin{cases} \frac{1}{j} & \text{if } \nexists C_k \in \mathcal{C} : \{\mathbf{x}, n_{\mathbf{x}}(j)\} \subset C_k \\ 0, & \text{otherwise} \end{cases}$$

where $\mathcal{C} = \{C_1, \ldots, C_K\}$ is the set of K clusters. Thus, higher-ranked (i.e. closer) nearest neighbours are penalized more severely for being assigned to a different cluster. Note that we use $j = 2$ so that the data point itself is excluded. In contrast with Handl and Knowles (2007), we use $\frac{1}{N}$ to take an average so that this measure can be compared across datasets. This has replaced the *overlap* as a problem feature as the connectivity provides a more detailed picture by incorporating more points than just the nearest neighbour.[2]

Number of clusters We simply use the true number of clusters (K), according to the labels. Datasets with larger numbers of clusters can potentially create challenges with the convergence of algorithms, and thus may provide some discriminative power.

[2]The *overlap* constraint is a special case of the connectivity, where $L = 2$ and a constant penalty of 1 is used.

Entropy of cluster sizes To provide a measure of the relative differences in size between the clusters, we calculate the entropy of the cluster sizes as follows:

$$H(\mathcal{C}) = -\sum_{k=1}^{K} |C_k| \; log_K |C_k|, \tag{6.6}$$

where $\mathcal{C}$ is the set of clusters, and $|C_k|$ is the (normalized[3]) size of cluster k. By using K as the base of the log, we ensure that for any K, equal distribution of cluster sizes will give $H(\mathcal{C}) = 1.0$, and $H(\mathcal{C}) \to 0$ in the other extreme (where one cluster has $N - K + 1$ data points). By ensuring that the resulting value is in the range $[0, 1]$, the calculated entropy is comparable across datasets of varying K and N.

Silhouette width standard deviation The average silhouette width across all samples (s_{all}) was previously used as a problem feature. Here, we use the standard deviation of the silhouette width (defined in Equation 2.4) across the samples, denoted s_σ, as a problem feature. This provides an indication of whether the clusters are equally well-assigned, or if there is a high degree of variation and thus a potentially "deceptive" silhouette width (see Figure 5.2 for an illustration of this). For completeness, this is calculated as follows:

$$s_\sigma = \sqrt{\frac{1}{N} \sum_{\mathbf{x} \in X} (s(\mathbf{x}) - s_{all})^2}. \tag{6.7}$$

Average eccentricity A notable omission from our problem features is a measure that captures the eccentricity of clusters. This is easy to calculate for HAWKS as we have the exact covariance matrix, but for other datasets this information needs to be extracted. There are various methods of either estimating the covariance matrix or measuring the eccentricity directly, but assessing which would be most *suitable* is non-trivial. We use a method similar to the calculation of eccentricity (Equation 5.5), but instead take an average of the largest to smallest eigenvalue ratios across all clusters. The eigenvalues accounting for 95% of the total sum are used, however, to avoid dividing by zero eigenvalues (which would occur with subspace clusters). As the resulting value is dimensionless, we can compare this across all datasets in this study.

[3]The cluster sizes are normalized such that they sum to 1 before calculation of the entropy.

Table 6.1: Dataset source parameters

Source	N	K	D	# Datasets
HAWKS	500, 2500	5, 30	2, 50	448
HK	5474 ± 2137	10, 20, 40, 60, 80, 100, 120	20, 50, 100, 150, 200	350
QJ	599 ± 289	3, 6, 9	5–24	243
SIPU	2248 ± 817	2, 15, 20, 35, 50	2, 4, 8, 16, 32, 64, 128, 256, 512, 1024	107
UCI	600 ± 571	2, 3, 4, 6, 7, 8, 10, 11, 15	3, 4, 6–11, 13, 18, 19, 22, 30, 34, 44, 60, 90, 166	20
UKC	30685 ± 2178	10, 11, 12	2	8

6.2.2 Additional datasets

In Section 5.2.4 we compared the datasets produced by HAWKS against two other generators: *HK* and *QJ*. In order to obtain further insights into the diversity of clustering datasets, we have added datasets from several other sources to see if they are in a different area of the instance space. Basic parameters about all sets of datasets can be found in Table 6.1.

***HK* (expanded)** We include the fuller set of datasets from the *HK* generator used in Garza-Fabre, Handl, and Knowles (2017). This set consists of 350 datasets from the "ellipsoidal" generator only, with 10 datasets each from 35 unique combinations of $K \in \{10, 20, 40, 60, 80, 100, 120\}$ and $D \in \{20, 50, 100, 150, 200\}$. Henceforth, *HK* will refer to this expanded set of datasets, which can be found at `https://github.com/garzafabre/Delta-MOCK`.

SIPU We use the clustering benchmark datasets used in Fränti and Sieranoja (2018). Specifically, we use the 'S-sets', 'A-sets', and 'G2 sets'. The 'S-sets' are all 2D data where $N = 5000$ and $K = 15$, but have different degrees of overlap between the clusters (using their definition of overlap, which is whether

the closest centroid is a different cluster). The 'A-sets' are also 2D data with varying numbers of (equally-sized) clusters. The 'G2 sets' of datasets consist of two Gaussians with varying degrees of overlap, constant size ($N = 2048$) and varying dimensionality (from 2 to 1024). Despite the lack of information about the underlying cluster generation mechanism, the variance of overlap and high number of dimensions presents some challenges for clustering algorithms. These datasets can be found at http://cs.joensuu.fi/sipu/datasets/.

UCI Machine Learning Repository The UCI machine learning repository (Dheeru and Karra Taniskidou 2017) is a popular database for datasets of varied domains and applications. Here, we use the same 20 datasets from this repository as was used in Arbelaitz et al. (2013), where the names and further details of each individual dataset can be found. These datasets were originally intended for supervised learning, but we assume that the labels represent clusters (which may be a poor assumption [Luxburg, Williamson, and Guyon 2012]).

UK Police Data These datasets were curated in Garza-Fabre, Handl, and Knowles (2017), and are the (anonymized) locations of crime. As such, all of these datasets are 2D, yet as illustrated in Figure 2.2 provide a variety of challenging cluster structures. Extreme outliers in the data have been removed to ensure that structure is present. These datasets are larger, with an average N of $\sim$30,000. As per their original source, these datasets will be referred to as *UKC*. More details about this data can be found in Garza-Fabre, Handl, and Knowles (2017) and its supplementary material, and the datasets themselves can be found at https://github.com/garzafabre/Delta-MOCK.

6.2.3 Experimental setup

To take a further step to evaluating HAWKS and its capabilities for producing a range of datasets more akin to a benchmark suite, we generate datasets using a wider range of parameters. The wider range of both datasets and problem features should enable a more robust comparison and evaluation of HAWKS. The experiment itself will be similar to Section 5.2.4 — the datasets will be run on the same set of clustering algorithms, and through their problem features be projected to 2D to create an instance space where we can evaluate their diversity.

Modifying the eccentricity constraint

The original constraint on eccentricity (Equation 5.5) in HAWKS measured the ratio of the largest to smallest eigenvalue of the covariance matrix, and then took the maximum of these across the clusters. An issue with this approach, is that it does not necessarily reflect the overall challenge that the dataset represents. For example, if we have two datasets that have one equally eccentric cluster, but in one dataset the remaining clusters are slightly less eccentric and in the other they are perfectly spherical, the resulting λ^{ratio} is still the same.

To put more pressure on HAWKS to generate datasets that are more wholly eccentric, we modify the constraint proposed in Equation 5.5 to the following:

$$\lambda^{ratio} = \min_{\forall\, k \in \{1,\dots,K\}} \frac{|\lambda_{\max}(\mathbf{\Sigma}^{(k)})|}{|\lambda_{\min}(\mathbf{\Sigma}^{(k)})|}, \tag{6.8}$$

such that we now use the *least* eccentric cluster to calculate the penalty. This enables the constraint to be more useful at embedding a preference of eccentricity when generating datasets.

HAWKS configuration

For this experiment, we use 64 unique combinations of parameters. To avoid flooding the instance space (and obscuring the other datasets) with datasets from HAWKS, we generate 7 instances from each of these combinations to produce 448 datasets (as shown in Table 6.1). In addition to the parameters shown there, we set $s_{\text{target}} \in \{0.45, 0.9\}$, two upper thresholds of the *overlap* constraint ($\{0, 0.1\}$), and two lower thresholds of the eccentricity constraint λ^{ratio} ($\{1, 50\}$). For the *overlap* constraint we have one setting that penalizes any, but the second setting allows for some overlap to create greater diversity (though at 10% the amount of *overlap* allowed is still small to ensure that the dataset has structure). For the λ^{ratio} constraint, our first setting allows any amount of eccentricity, but the second tries to enforce eccentricity (which is now taken to be the minimum across all clusters). To ensure a balanced trade-off between the different values of s_{target} and the constraints, we keep $P_f = 0.5$. The full configuration can be found online[4].

[4] https://github.com/sea-shunned/thesis_material/chp6_exploring.json

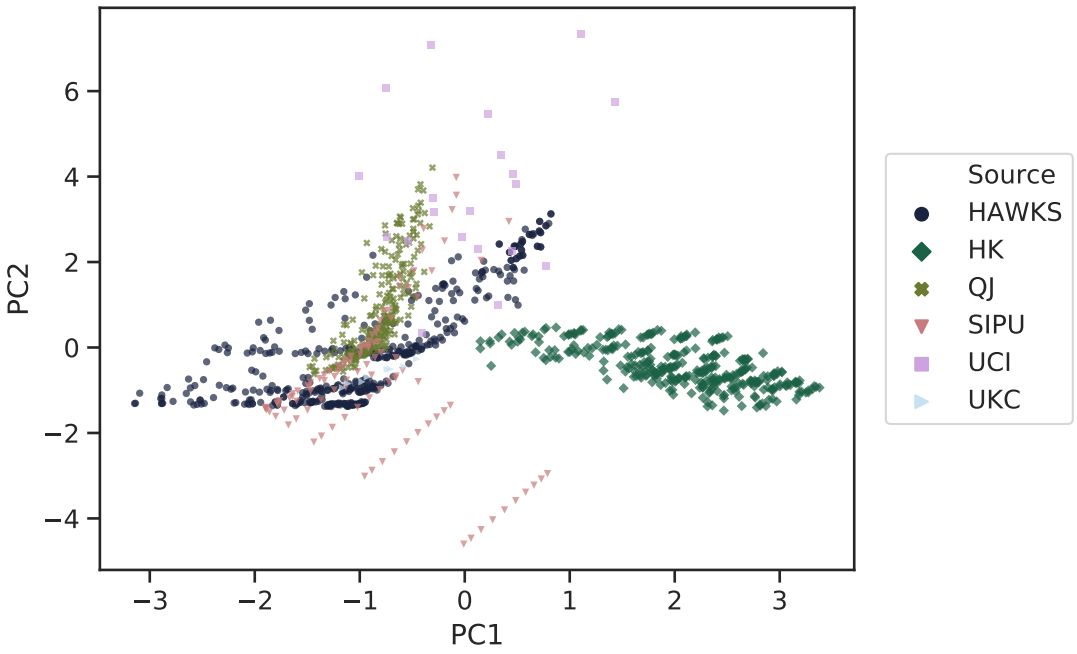


Figure 6.8: The instance space of the datasets with each of the six dataset sources highlighted.

6.2.4 Results

Figure 6.8 shows the instance space, with the different sources of the datasets highlighted. The variance contribution of the different problem features in this space is shown in Figure 6.9, where the magnitude of the line is correlated with the variance contribution to show both in which direction of the space that feature varies, and its relative contribution. For example, we can see that the connectivity feature was quite distinct when compared to the entropy or number of clusters. To see how each problem feature varies across the instance space, the full set of figures can be found in Section A.5.1. The two principal components account for 57.56% of the variance, and so there is some information loss in this projection.

The spread of datasets generated by HAWKS is highly encouraging, particularly as (in contrast to the feature set of Section 5.2.4) the problem features are more decoupled from HAWKS' parameters. As was previously seen in Section 5.2.4, the *QJ* generator is also well spread across the space, though this time as a narrow band across the space. The expanded *HK* datasets spread in a noticeably different direction to the other datasets. Looking at the problem feature contributions and from knowledge of the generator itself, this is likely

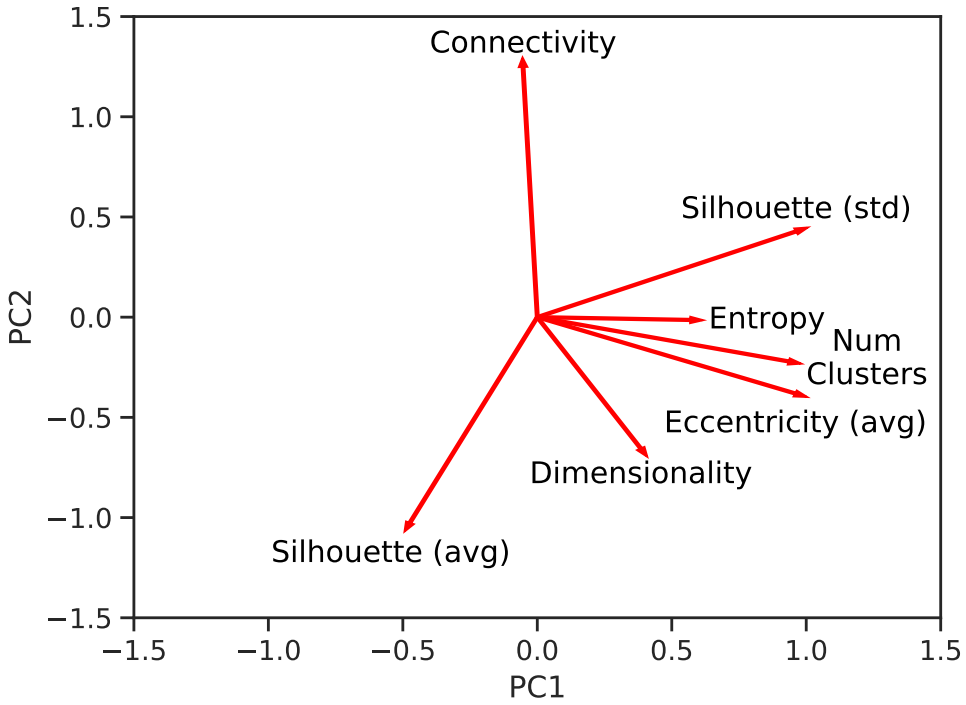


Figure 6.9: The direction and magnitude of the problem features as they contribute to the first two principal components.

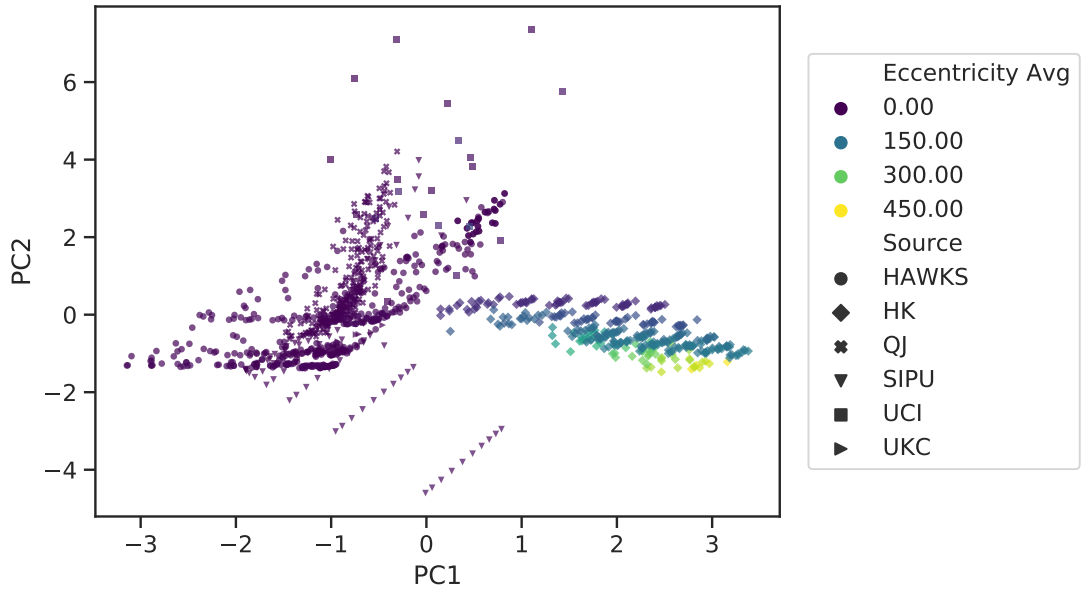


Figure 6.10: The instance space colourized by the value of the average eccentricity problem feature, indicating that a subset of the *HK* datasets are significantly more eccentric than any of the other datasets.

due to the higher number of clusters and more eccentric clusters. This generator was designed specifically for creating eccentric clusters in higher dimensions, which when we look at the average eccentricity feature (in Figure 6.10) it is very successful at. The diversity of performance on datasets from this generator was not reflected in a spread across the space in Section 5.2.4, so the spread seen here indicates that our greater set of problem features is significantly better at capturing different challenges or properties of the datasets.

The *SIPU* datasets show a strong banding of instances, particularly due to the 'G2 sets' which have a much higher dimensionality than the other datasets we use. This implies that there is low variation among the datasets from each configuration. The *UCI* datasets are very spread across the space, though this is unfortunately due to a lack of structure (which we later explore when looking at clustering algorithm performance), as their higher connectivity indicates that the labels do not line up with a locality perspective of clustering. Finally, the *UKC* datasets do not seem to represent anything extraordinary with regards to the problem features we use here, leading to the conclusion that either the synthetic datasets used here are not too dissimilar to real-world data or our set of problem features does not capture some aspect of complexity that they uniquely exhibit — as we see in the following section, these datasets seem to pose no greater difficulty for clustering algorithms than any other used here, so it is likely the former.

The more stochastic nature of HAWKS is illustrated by its lack of banding, as multiple datasets from the same configuration can result in quite different datasets depending on the trade-off that was found, rather than a dataset with very similar properties (but just a different location of the clusters). Nevertheless, the clear separation between the datasets of HAWKS and *HK* either speaks to a limitation of HAWKS, or a lack of diversity in the parameterization. We investigate this further in Section 6.2.5, after looking at how these problem feature differences corresponded with a difference in cluster algorithm performance.

Clustering algorithm performance

The aggregated performance of the clustering algorithms across all datasets from each source is shown in Figure 6.11. We can see a varied tale of performance across the sets, some of which was observed from the instance space. HAWKS appears to exhibit the broadest spread of performance across all algorithms. The *HK* generator seems to have generated datasets that were generally very difficult for

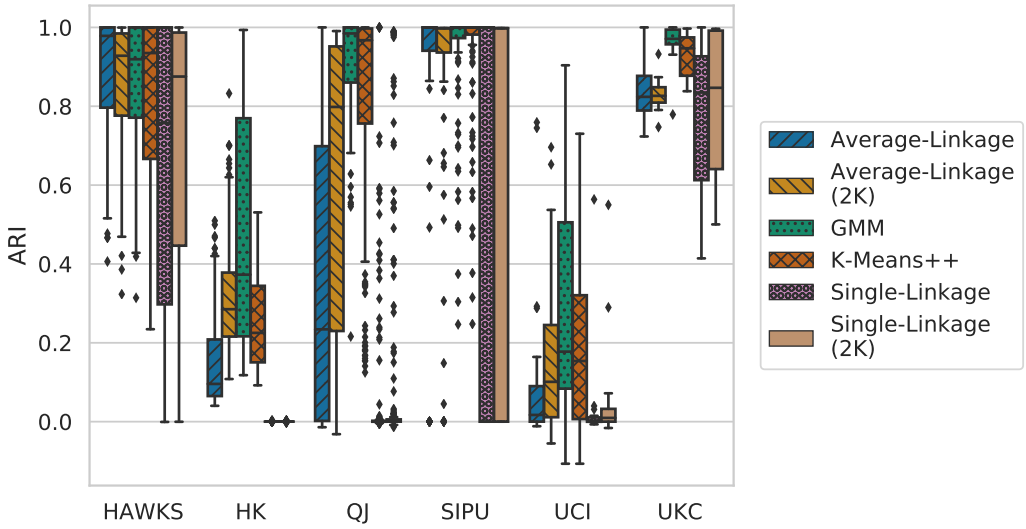


Figure 6.11: Clustering performance (ARI) for each algorithm on each of the dataset sources. The size (IQR) of the boxplot indicates diversity of difficulty of the datasets, and the median indicates the average difficulty for that algorithm.

the algorithms, with a low mean ARI for each (and incredibly poor performance for single-linkage). As this generator is explicitly designed to create eccentric clusters in high dimensions, the eccentricity of these clusters may in-part explain these results. Looking at Figure 6.10[5], we can see that in general these datasets are considerably more eccentric than those from any other source.

The poor structure of the *UCI* datasets is reflected in the poor performance obtained by all clustering algorithms, even obtaining negative ARI values. The *SIPU* datasets, however, exhibit a very different view, where the datasets are generally very easy for average-linkage, K-Means++, and GMM, but have a very wide spread of performance for single-linkage (due to the overlap). Interestingly, for many of the datasets the ARI for single-linkage is either 0 or 1, though the median here is ≈ 1.0, highlighting the sensitivity of this algorithm where close clusters are chained together (Hubert 1974). The *UKC* datasets do not vary significantly in the challenge that they pose, with all algorithms generally doing well (though again the sensitivity of single-linkage is highlighted with its higher range of ARI values obtained).

The critical difference (CD) diagrams previously shown in Figure 5.13 are shown for this experiment in Figure 6.12. Notably for HAWKS, average-linkage

[5]Visualizations of the instance space for each of the other problem features can be found in Figure A.3.

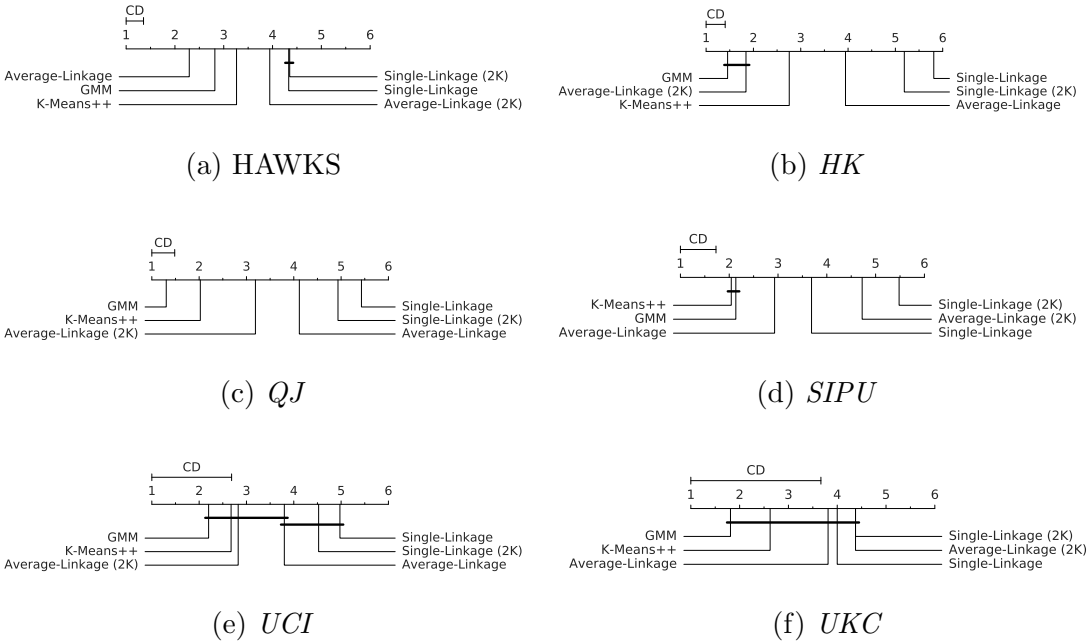


Figure 6.12: Critical difference (CD) diagrams for each of the dataset sources, showing the average rank for each algorithm across the datasets. Solid lines connect algorithms which are not significantly different from each other according to a two-tailed Nemenyi test.

was the best performing algorithm on the datasets produced, owing to the greater diversity of clusters that are less obviously Gaussian. GMM, as before, performed significantly better than K-Means++, which in turn was superior to the remaining linkage-based algorithms. Single-linkage achieved the same rank even with an additional number of clusters, indicating that the cause of poor performance was either not due to the chaining effect, or an even higher number of clusters was needed (which could be caused by tightly-packed eccentric clusters in higher dimensions).

GMM and average-linkage (with $2K$) were equally superior to the other datasets for the *HK* datasets, likely due to their high eccentricity. The additional clusters for average-linkage helped to avoid the assignment of multiple clusters into a single one, possibly caused due to highly eccentric clusters in close proximity in higher dimensions, as evidenced by the location of the datasets in the instance space where the $2K$ variant was superior (shown in Figure 6.13). The boxplots for these algorithms (Figure 6.11) indicate that for some datasets, GMM was able to perfectly cluster the data, but for the majority of datasets it was not

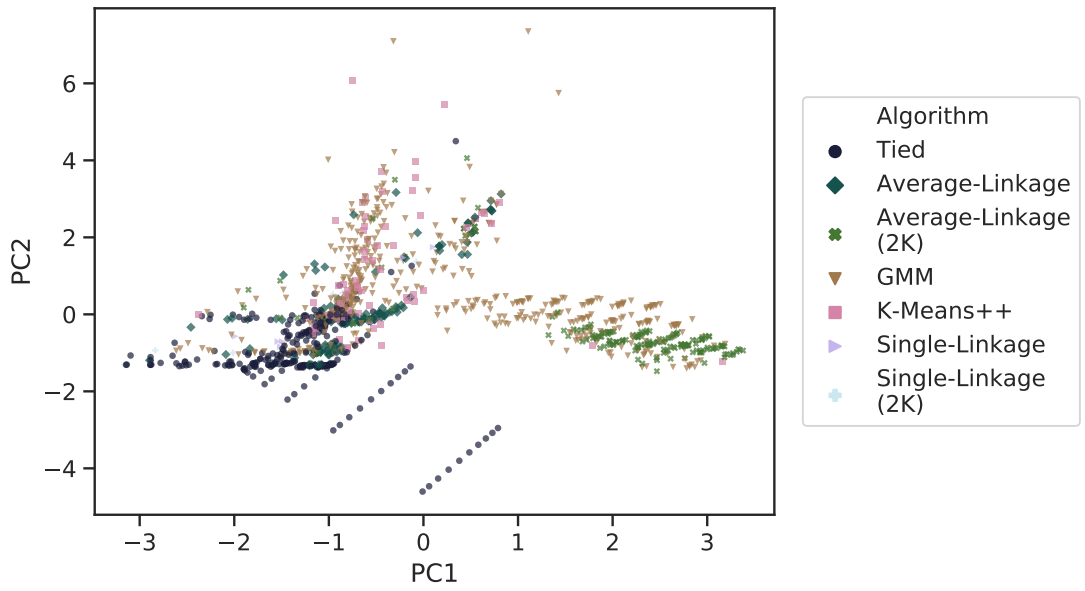


Figure 6.13: The instance space, where the algorithm that performed the best (highest ARI) is highlighted.

able to. The high dimensionality of these datasets impedes visualization to elucidate further reasons for this low performance, though the distinguishing feature and likely reason is the eccentricity, and the higher standard deviation of the silhouette width combined with lower average silhouette width implies a lack of separation between the clusters (shown in Figure A.3).

Two groups of algorithms were found for the *UCI* datasets (Figure 6.12e), whereas no algorithm was superior to any other for the *UKC* datasets (Figure 6.12f). The poor structure in the former datasets makes it harder for algorithms to differentiate performance, and the smaller number (8) of datasets in the latter is likely not enough to tease out significant differences between the algorithms. The higher connectivity in the *UCI* datasets (shown in Figure A.3b) can also explain the lower ranking of single-linkage. The ranks of the algorithms in Figure 6.12f favour the compactness-based algorithms (GMM and K-Means++), which is somewhat expected when visualizing the datasets (one of eight *UKC* datasets was shown in Figure 2.2, though visualizations of all can be found in the supplementary material of Garza-Fabre, Handl, and Knowles [2017]).

Considering algorithmic "footprints" in the instance space, we can see the space highlighted according to the algorithm that achieved the best ARI in Figure 6.13. Clearly there are some areas where particular algorithms dominate,

Table 6.2: Number of 'wins' (highest ARI for a given dataset) for each algorithm from each of the sources.

Source	Average-Linkage	Average-Linkage ($2K$)	GMM	K-Means++	Single-Linkage	Single-Linkage ($2K$)	Tied
HAWKS	117	25	107	25	7	5	162
HK	–	149	199	2	–	–	–
QJ	2	5	157	20	–	1	58
SIPU	–	–	11	21	–	–	75
UCI	2	3	10	4	–	–	1
UKC	–	–	5	1	–	–	2

indicating that the expanded set of problem features are more discriminative and capable of separating the datasets. The 'tied' datasets are those on which at least two algorithms achieved the same score (i.e. a perfect ARI of 1), indicating that these are trivial datasets to solve. For further detail, in Table 6.2 we provide a table of counts for each algorithm broken down by each dataset source. We can see that HAWKS was the only generator both to produce datasets where every algorithm performed the best, and capable of producing datasets specifically where single-linkage performed the best.

Overall, as there are not clear boundaries of where particular algorithms are dominant, this may hint that either the projection method results in too much information loss, or that our problem feature set still needs further refinement to capture a broader range of properties that cause differential performance.

6.2.5 Targetting the instance space

We previously discussed (Section 4.3.3) the generation of datasets in a specific area of the instance space, an example of which can be found in Smith-Miles and Bowly (2015). Given that we have an instance space, we can identify regions where we would like datasets with those properties but do not currently have any. There are gaps in our instance space where datasets either cannot exist (e.g. high silhouette width and high connectivity), it would be undesirable to generate them (e.g. high silhouette width variance), or it is less straightforward

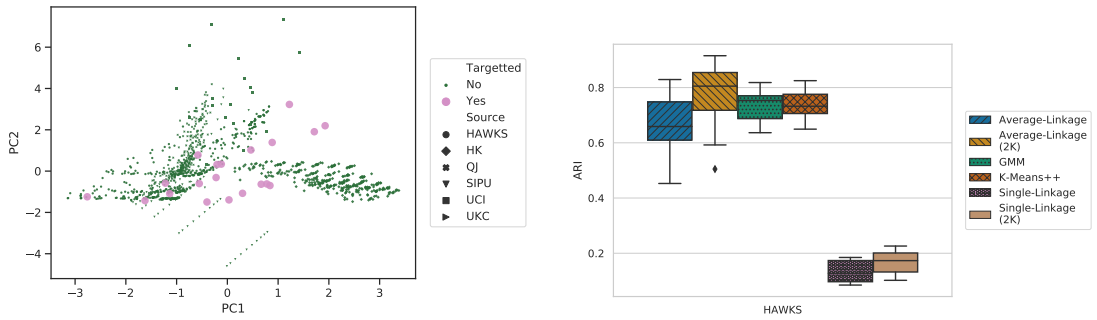
how to generate such datasets (e.g. high silhouette width variance). As such, a downside is that some understanding of the features and their relation is required to identify *meaningful* gaps.

Nevertheless, in this section we identify a region of the instance space with no datasets from HAWKS and identify a set of parameters that may produce datasets with such problem feature values. Unlike Smith-Miles and Bowly (2015), we do not explicitly incorporate information about the instance space into the generating mechanism — all we have is the parameters that HAWKS provides. While this is useful for generating datasets from a user-perspective, by not incorporating a target (informed by the instance space) into the generating mechanism, we may limit our ability to generate datasets in a target region of the space. This is particularly true as the link between the parameters and the problem features may not be obvious (e.g. the standard deviation of the silhouette width is influenced by multiple parameters simultaneously). As explored in Smith-Miles and Bowly (2015), explicit targeting of the instance space would require a modification of the objective function to reflect proximity to this target, which in our case would result in a loss of the aforementioned advantage in having a user-defined parameter for the 'difficulty'.

Eccentric clusters

From the average eccentricity feature values obtained in particular by the *HK* datasets (shown in Figure 6.10), it's clear that the parameterization of the λ^{ratio} constraint has been conservative. As such, we run HAWKS another 10 times with parameter modifications. We set $\lambda^{ratio} \geq 100{,}000$, and the upper bound of the range used for sampling the initial cluster eigenvalues (defined in Equation 5.3 as β_2, which has been hitherto unchanged) is increased significantly ($\times 1{,}000$) to try and create more eccentric clusters from the start. To further push HAWKS towards generating eccentric clusters, we set $P_f = 0.25$, putting more emphasis on satisfying the constraints. For the fitness, $s_{\text{target}} = 0.6$ is used to help encourage bringing the clusters together so that the eccentricity is more impactful (for very well-separated clusters, high eccentricity is less challenging). Similar to the *HK* datasets, any *overlap* is penalized. We also use a larger number of smaller clusters ($K = 100$) and data points ($N = 4000$) to better emulate the *HK* datasets. The full configuration can be found online[6].

[6] https://github.com/sea-shunned/thesis_material/chp6_targetting.json



(a) The generated datasets in the instance space (highlighted by larger marker size)

(b) Algorithm performance for the generated datasets

Figure 6.14: The generated datasets visualized in the instance space (a) and the performance of the clustering algorithms on these datasets (b).

Figure 6.14 shows the 20 generated datasets in the instance space (the larger dots are the generated datasets). The majority of the datasets are in areas that HAWKS has not previously explored, several of which are closer to the *HK* datasets as desired. Two of these datasets in particular are in regions of the instance space distinct from all of the other datasets, implying that these datasets have quite separate properties (from inspection of the problem feature values, the primary difference is a higher connectivity). The performance of the clustering algorithms on these datasets, shown in Figure 6.14b, indicate that (at least for average-linkage) the large distance between these datasets in the instance space is reflected in a variance of performance. Similar to the results seen in Table 6.2, the higher performance of average-linkage ($2K$) indicates a greater similarity between these new datasets and the *HK* datasets.

6.2.6 Experimental summary

In this section, we ran HAWKS with a wider set of parameters, creating an instance space with a broader range of problem features (that were not directly encoded by HAWKS), comparing against more varied sets of datasets. The set of parameters helped HAWKS generate more diverse datasets, resulting in a more uniform spread of performance across several clustering algorithms than the other datasets. The additional problem features were useful in creating a more informative instance space, though the quest to find additional (uncorrelated) features is a constant source for improvement. We were able to identify regions of the space without datasets to then more explicitly generate datasets from HAWKS with

those problem feature values. The more stochastic nature of HAWKS results in a wider spread of these datasets across the space than may be desirable when trying to explicitly generate datasets in a certain region, though additional tweaking of HAWKS' parameters may prove beneficial in such a scenario.

6.3 Maximizing performance difference

In our two larger experiments (Section 5.2.4 and Section 6.2), we saw that different generators tended to generate datasets that favoured particular algorithms (in terms of the ease of getting an ARI close to 1). Our expanded set of problem features helped somewhat to capture these differences, but to do this accurately we require a *full* feature set ($\mathcal{F}$) that describes all aspects of a problem, which as we have discussed is particularly challenging (if even possible) with clustering. A potential way to circumvent the recondite and potentially unquantifiable features of the data that present different challenges for clustering algorithms is to instead generate datasets that directly maximize the performance difference between them. The cluster structures that favour one algorithm over another can then be discovered without explicit modelling or *a priori* knowledge of these structures.

This facilitates obtaining better insights into the strengths and weaknesses of algorithms, and in particular to get a deeper understanding of the *relative* differences between algorithms. This goes beyond the original ASP (Section 4.2), which is concerned with predicting the best clustering algorithm for a dataset based on its problem features. The approach proposed in this section is more focussed on directly comparing clustering algorithms in order to obtain insights about their performance, which can aid in the more fundamental development of clustering approaches.

To enable HAWKS to do this, our objective needs reformulation in order to capture the performance difference between two algorithms:

$$\max\ f(X) \equiv \max\ \phi(\alpha_w) - \phi(\alpha_l), \tag{6.9}$$

where $f(X)$ is the fitness function (the original fitness function was defined in Equation 5.4) applied to dataset X, ϕ is a scoring function (e.g. the ARI), α_w is the *winning* algorithm we want to perform better relative to the *losing* algorithm,

α_l. This creates a new mode for HAWKS, named 'versus' mode, as we create an adversarial[7] situation between the two clustering algorithms.

Otherwise, HAWKS remains the same as in the previous section. As our fitness function no longer relies on the silhouette width, in this mode HAWKS cannot use the PSO-informed mutation operator. With no explicit control over the silhouette width, there is the potential to generate datasets with low s_{all} (and thus are uninteresting datasets due to their poor structure), but as the difference between the algorithms is being maximized this is unlikely to occur due to a poor structure being generally 'unclusterable' by all algorithms.

6.3.1 Experimental setup

To ascertain the capabilities of HAWKS in generating datasets that challenge different algorithms differently, we run each of the four previously-used algorithms against every other in a head-to-head. These different combinations will present varied difficulty to the optimization (based on the perceived similarity of the algorithms). The datasets produced will then be run on all clustering algorithms to get a broader view of the challenges that these datasets propose.

Once the ability of HAWKS to generate datasets in this setting has been ascertained, we pose the situation where HAWKS is unable to generate datasets that favour one algorithm over another. In such a scenario, it can be difficult to conclude whether this is due to the superiority of one algorithm over another, or the inability of HAWKS to generate structures with properties that *would* differentiate them. Of additional interest is *how* these datasets are generated — for example, if datasets are generated to fool one algorithm through exploiting highly overlapping clusters, while this presents an insight into the different levels of robustness to overlap, it does not provide datasets that are generally useful due to their poor structure.

Despite HAWKS' utility for embedding a preference towards either the objective or the constraints, in previous experiments we were more interested in generating a diversity of datasets and so we kept $P_f = 0.5$. For this 'versus' mode, it is clear that some pairings of algorithms are likely to be challenging for HAWKS. In such a scenario, while we still have constraints that are important,

[7]According to the non-scientific definition, as opposed to the field of adversarial machine learning. Although the algorithms are against each other, there is no communication between them and their input is the same.

we wish to avoid situations where the optimization is being mainly driven by reducing the constraint penalty. Rather than simply removing the constraints, we can increase P_f to add further emphasis in the optimization on producing a difference in algorithmic performance. For generating datasets that cause a maximal difference between e.g. K-Means++ vs. GMM, such pressure may be needed. Note that, "K-Means++ vs. GMM" refers to K-Means++ as the *winning* algorithm (α_w), and GMM as the *losing* one (α_l). Without *a priori* knowledge of the difficulty, this provides a useful lever for driving dataset generation through iteration and interaction with HAWKS. To encourage a performance difference, we use $P_f = 0.75$ in all experiments, though this can be adjusted if the ARI difference is achieved through e.g. too much violation of the *overlap* constraint.

For our scoring function (ϕ) we use the ARI to provide a comparable measure of agreement with the true labels of the datasets. While any scoring function can be used here for the optimization, an unbiased external validity index provides a broader perspective of performance and thus can be used with any clustering algorithms with no preference of structure.

HAWKS configuration

This set of experiments was performed using a single configuration, which can be found online[8]. We use the PSO-random mutation operator in this experiment, with $overlap \leq 0$ (fully constrained) and $\lambda^{ratio} \geq 1$ (fully unconstrained). For visualization purposes, we generate 2D data with $K = 5$ clusters and $N = 2000$.

Avoiding a completely overlapping initialization of the clusters is also likely to help speed convergence and create more potentially interesting structures, as regardless of properties such as eccentricity if the clusters are well-separated algorithms are likely to do equally well. As such, we use the same setting as the 'apart' initialization scenario used in Section 6.1.2.

6.3.2 Results

To get an overall impression of the ability of HAWKS to successfully optimize the differences between the algorithms, we refer to Figure 6.15. Each non-diagonal plot in this grid (i.e. the line plots) represents the best ARI difference (i.e. highest fitness) for each of the 30 runs between the α_w (the algorithm for that row) and

[8] https://github.com/sea-shunned/thesis_material/chp6_versus.json

the α_l (column). For example, average-linkage vs. GMM is the second plot in the top row. We can visually identify the ability of HAWKS to create a difference by observing the angle of the lines. For each plot on the diagonal, we have the overall mean (and standard deviation) of the ARIs achieved when that algorithm was chosen as the winner (on the left-hand side) and as the loser (on the right-hand side). This illustrates the ability of HAWKS to more *generally* produce datasets that the particular algorithm finds either easy or difficult to solve. We can observe that for K-Means++, HAWKS has the flexibility to produce datasets that are both easy and difficult, more so than when compared to single-linkage where it is significantly easier to generate datasets it performs poorly on, and more difficult to generate datasets it can uniquely perform well on.

For the stochastic algorithms (K-Means++ and GMM) there is inherent variability in the fitness evaluation[9]. As each parent is evaluated once, despite our use of multiple initializations for these algorithms there remains the inevitable possibility that the best resulting model from these initializations is only locally optimal, and thus may give a deceptively low or high fitness value. To illustrate to what extent this affects the results, we run the clustering algorithms again (using the same parameters) with a different set of initializations. In Figure 6.16, we can see overlaid on each plot a new set of 30 lines with the updated ARI values, as well as the updated aggregate on the relevant diagonal plot. Note that for ease of illustration we have skipped the two scenarios where only deterministic algorithms were involved (average-linkage vs. single-linkage and vice versa). Most notably, we can see the performance for GMM increase for many runs when it was the *losing* algorithm, indicating that for these datasets GMM was capable of doing better — across all runs as α_l, GMM saw an increase of $\simeq 0.14$ in the mean ARI. Of course, the stochastic nature of these algorithms is an inherent disadvantage, and giving them an unrealistic computational budget would not reflect this natural disadvantage of these algorithms over e.g. linkage-based algorithms.

In the following sub-sections, we go into further detail on particular combinations of algorithms, providing examples of datasets that were generated. In the example datasets we show, a difference in colour/marker indicates a different assignment of cluster. For each example, the performance (ARI) is shown above the dataset. It is not reasonable to go through every combination here, so

[9]Interestingly, when the initialization of these algorithms did not change between evaluations, HAWKS moved clusters such that the initialization (when designated as the *losing* algorithm) would split them into multiple clusters, thus reducing performance in an undesirable way.

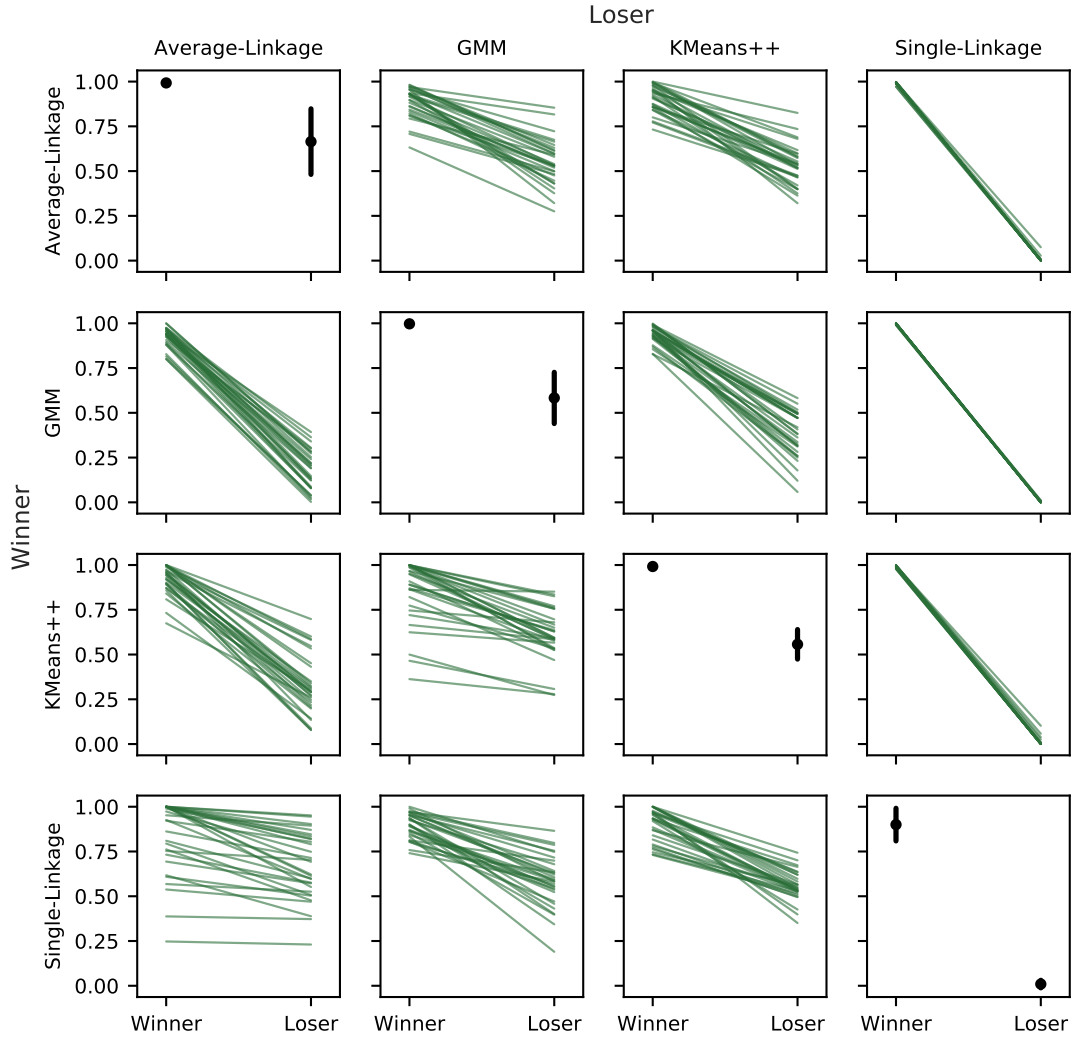


Figure 6.15: Grid of plots showing the ARIs for each algorithm as both the α_w (rows) and α_l (columns). Each line represents the best individual found from a single run (across 30 runs). The angle of the line indicates the performance difference, and the spread of the lines indicate robustness. On the diagonal is the aggregated mean and standard deviation across all of the (best) individuals where that algorithm was the α_w (left-hand side) and α_l (right-hand side).

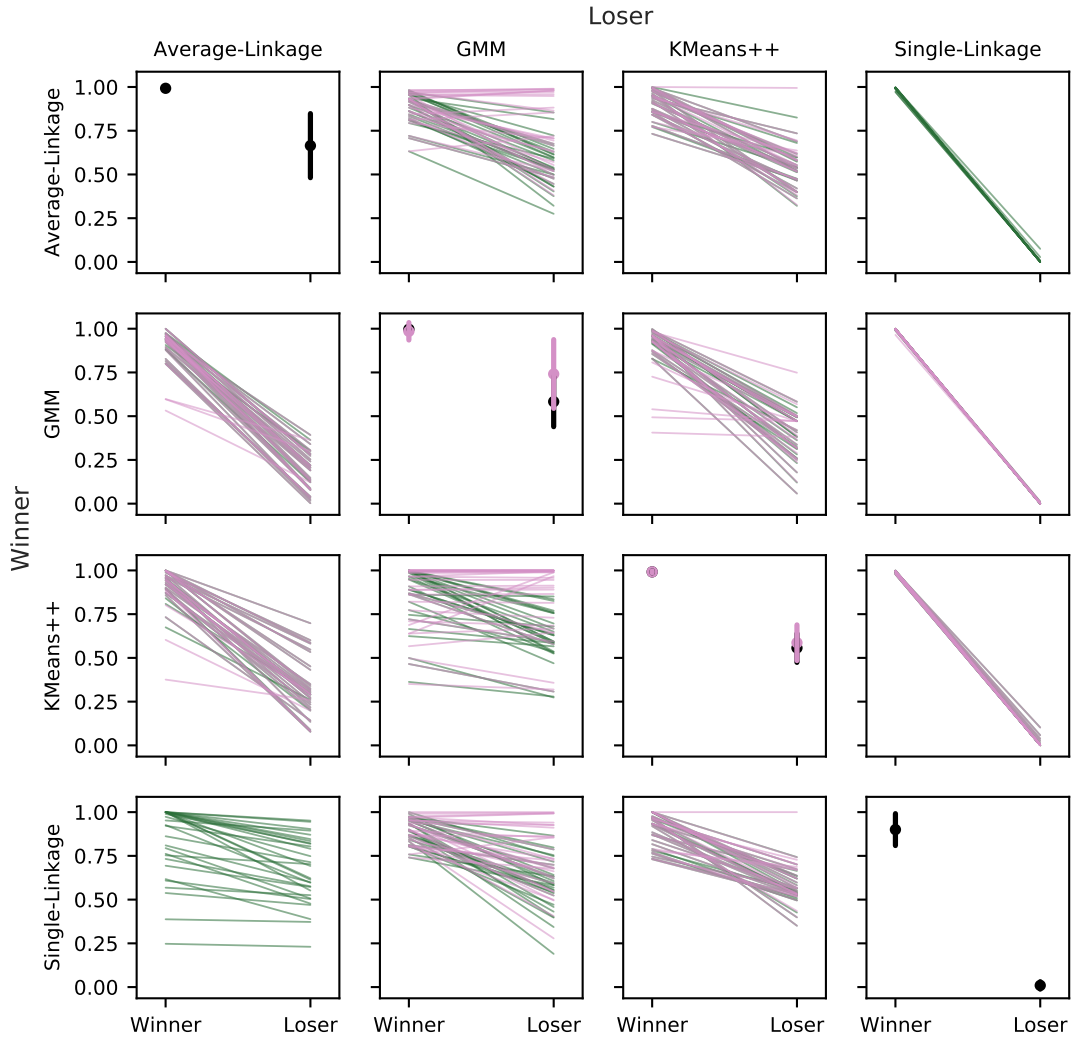
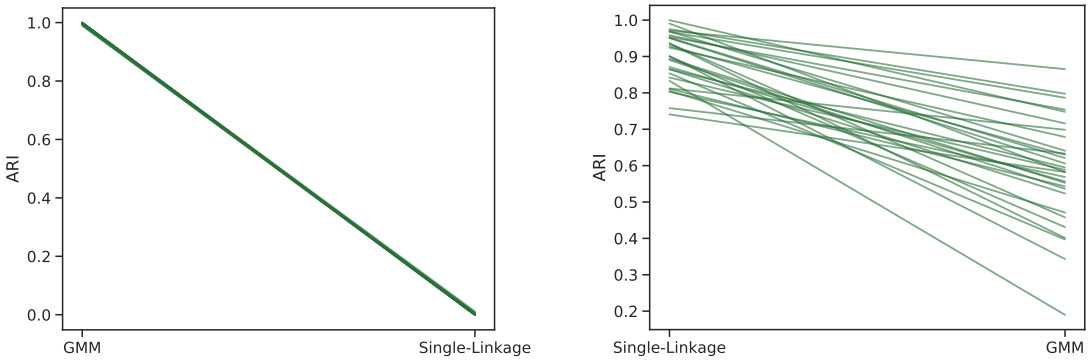


Figure 6.16: Same plots as Figure 6.15, but with each of the clustering algorithms (combinations with only deterministic algorithms have been skipped) run again with different initializations to ascertain the stochasticity in the obtained ARI scores. The aggregated range of ARIs obtained with the new initializations is also shown in the diagonal plots.



(a) GMM vs. single-linkage

(b) Single-linkage vs. GMM

Figure 6.17: The ARIs obtained across the 30 runs for each of the head-to-heads between GMM and single-linkage.

we select a few combinations that cover all of the algorithms, illustrating varied insights and (to some degree) the success of this mode.

GMM and Single-Linkage

In Section 5.2.4 all three generators produced datasets that, on average, GMM performed better on compared to single-linkage. This difference is inescapably due to the convex clusters that all these generators create. The results in Section 6.2.4, however, indicated that HAWKS does not generate datasets that are all easily solved by GMM only, and that it can create datasets in which single-linkage performs the best. While we expect HAWKS to easily maximize the difference of GMM vs. single-linkage, the reverse scenario should be more difficult.

Figure 6.17 compares the ARI for GMM and single-linkage side-by-side for each of the 30 runs. When α_w = GMM (Figure 6.17a), there is a very clear trend towards an ARI difference of 1, showing that HAWKS was consistently able to generate datasets that create a very large performance difference between the two algorithms. In Figure 6.17b, we see a more varied picture. The spread of ARI performance for single-linkage is much higher, indicating that HAWKS was not always capable of producing datasets that single-linkage could perfectly solve. The average angle of the lines suggest that a consistent performance differential could be created, but this difference is much lower than in the reverse scenario.

Although this result is not unexpected, by looking at the datasets produced we can investigate *how* the clusters were placed to create this large performance difference. In Figure 6.18 we can see an example dataset with an ARI difference

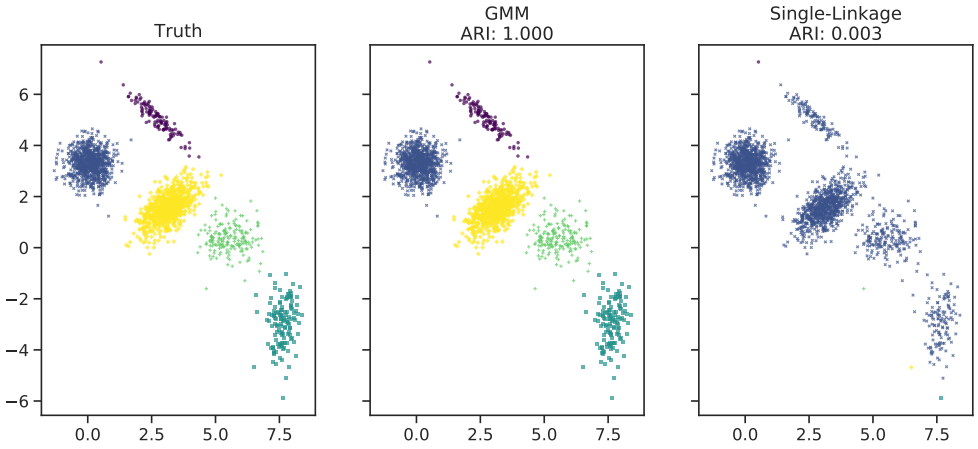
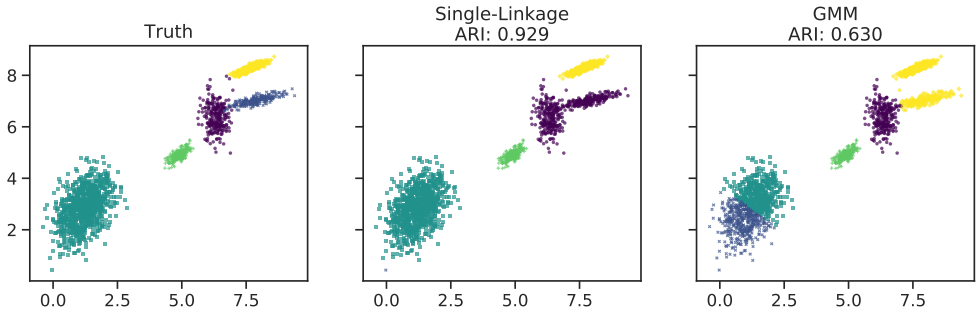


Figure 6.18: Example dataset from GMM vs. single-linkage, illustrating that placing clusters in close proximity to each other easily resulted in single-linkage chaining the clusters together.

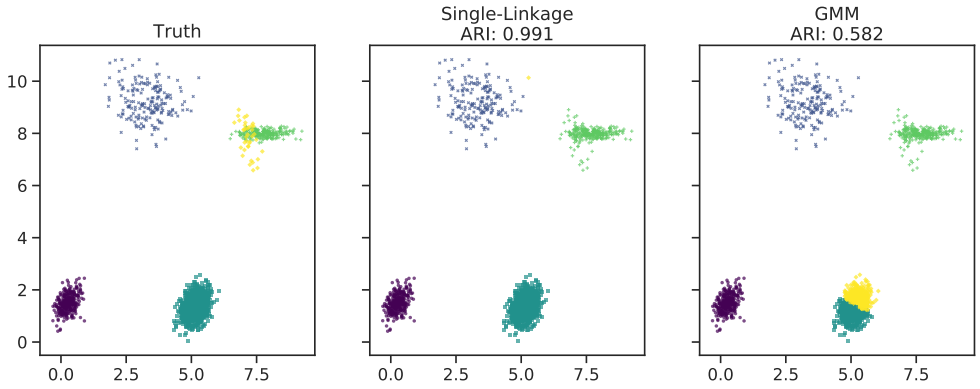
of $\simeq 1.0$. By placing the clusters close together, the few datasets that are sampled further away from the mean induce the 'chaining effect' of single-linkage (Hubert 1974) such that the *vast* majority of points are assigned to a single cluster, yet these clusters are still separated enough that GMM can precisely identify the correct membership.

Figure 6.16 showed that, for a significant number of runs, there was an improvement in performance of GMM when run again with a different initialization, highlighting the cause of the lower ARI. As previously discussed, the poor convergence due to poor initialization is an inherent limitation of this algorithm, as simply running the algorithm more times is not always a practical solution. By inspecting the datasets produced, there were two main ways that HAWKS seems to generate datasets where single-linkage could perform better: move larger clusters out far from each other, and overlap smaller clusters such that GMM splits larger clusters. Of course, the latter of these is not necessarily useful (from the perspective of generating realistic datasets), but both of these approaches appear to exploit the dependence of GMM on initialization. Figure 6.19 shows two examples of these exploitations. The degree of overlap in both examples is small, particularly in Figure 6.19a where $overlap = 0.004$, and so these are not implausible datasets.

Of further interest is the contrast of performance across all algorithms on the datasets produced in these two scenarios. In Figure 6.20a we can see that most algorithms did well apart from single-linkage, showing that the close proximity of

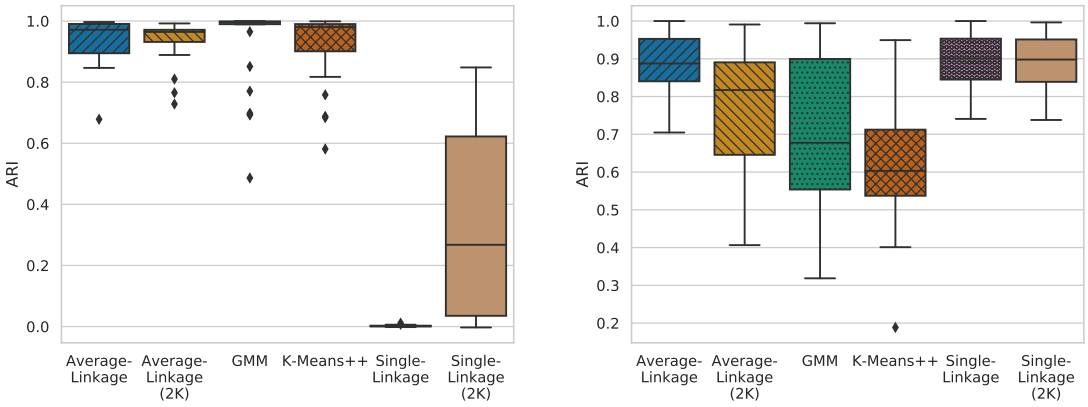


(a)



(b)

Figure 6.19: Two example datasets from single-linkage vs. GMM, highlighting the exploitation of overlap and well-separated clusters to induce poor initialization and thus local optima.



(a) GMM vs. single-linkage

(b) Single-linkage vs. GMM

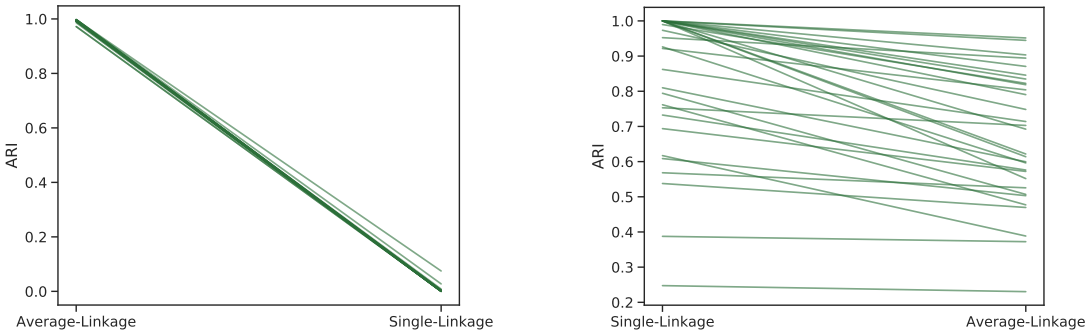
Figure 6.20: Clustering performance across all algorithms for the two head-to-head scenarios between GMM & single-linkage.

clusters was a challenge unique to single-linkage. The variant that used double the number of clusters ($2K$) showed a significant improvement, highlighting the need for this algorithm to have an *over*-estimated K for less-compact clusters in close proximity. The placement of the clusters that resulted in poor GMM performance generally resulted in even worse performance for K-Means++ (shown in Figure 6.20b), highlighting their shared weakness. The improved initialization scheme of K-Means++ should have somewhat helped avoid large clusters being split, but the close proximity and eccentric clusters (as highlighted in the example datasets) are a larger barrier to convergence (compared to GMM). Average-linkage was able to perform almost as well as single-linkage, as expected.

Average-Linkage and Single-Linkage

Figure 6.15 showed two notably different profiles of performance between the head-to-heads of these algorithms, worthy of further investigation. The first scenario, average-linkage vs. single-linkage, displays very similar results to GMM vs. single-linkage, in terms of both the ARI difference (Figure 6.21a) and exploitation used (Figure 6.22). By taking an average of the distances between two groups, average-linkage is not susceptible to the chaining effect that single-linkage is (Hubert 1974; Yim and Ramdeen 2015), making it trivial to induce this effect.

We can see a very high spread of ARI differences for single-linkage vs. average-linkage (Figure 6.21b). There are some datasets that correlate both high and low



(a) Average-linkage vs. single-linkage (b) Single-linkage vs. average-linkage

Figure 6.21: The ARIs obtained across the 30 runs for each of the head-to-heads between average-linkage and single-linkage.

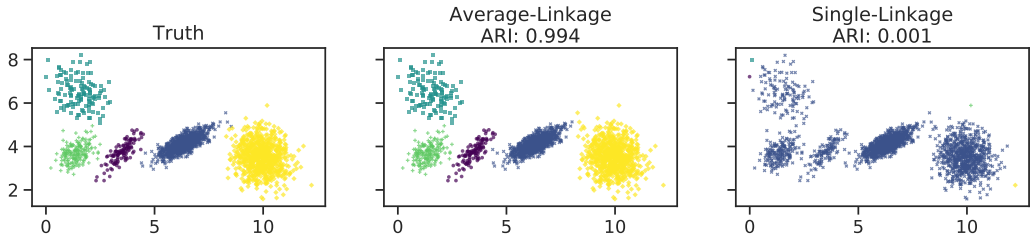


Figure 6.22: Example dataset from average-linkage vs. single-linkage, illustrating that placing clusters in close proximity to each other easily resulted in single-linkage chaining the clusters together (as seen previously with GMM vs. single-linkage).

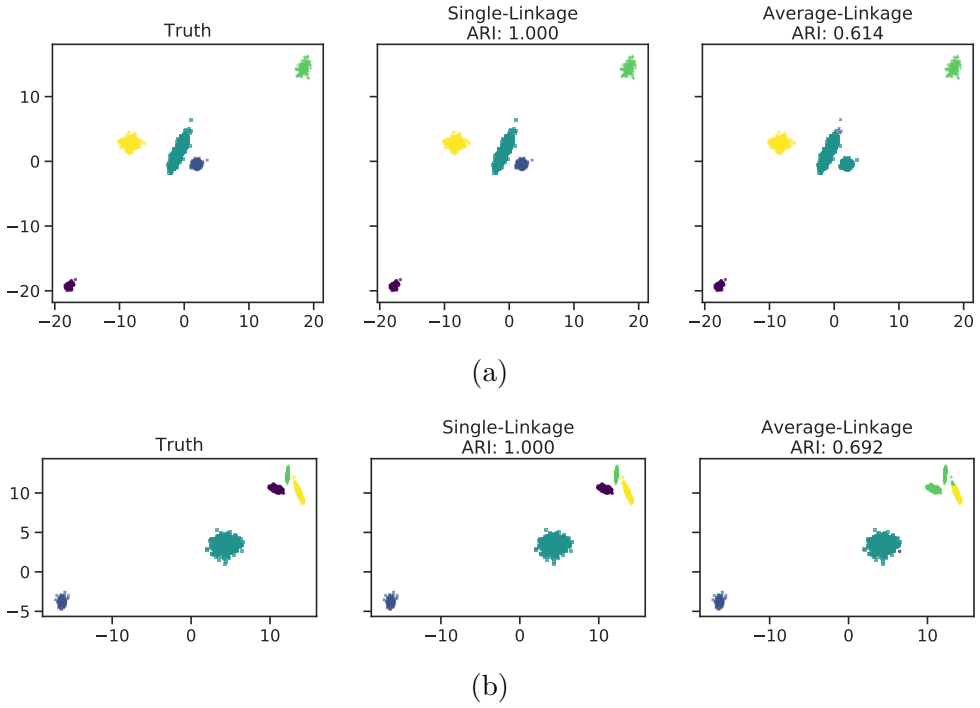


Figure 6.23: Two example datasets from single-linkage vs. average-linkage, highlighting the exploitation of separating most clusters to cause a split in clusters placed closer together.

performance between the two, indicating that the algorithms do share some similarities. In Figure 6.23, we can see two example datasets where the ARI difference was larger between the two (and maximized for single-linkage), indicating that HAWKS managed to find a structure that better favoured single-linkage. Examining these datasets, we can see that in both examples the clusters are in general well-separated. By placing some clusters much further away, and co-locating a few smaller clusters, the averaging criterion (to determine where to split next) used by average-linkage assigns clusters that are closer together the same cluster label. The mixed eccentricities of clusters that HAWKS is able to generate helps facilitate the discovery of this exploit.

We can also see that this co-location of several clusters (relative to the large distances between the well-separated clusters) is reflected in a degraded performance of the other clustering algorithms, as shown in Figure 6.24. A similar weakness was exploited in single-linkage vs. GMM, which GMM also shares with K-Means++. To explore the differences between these two algorithms, in the next

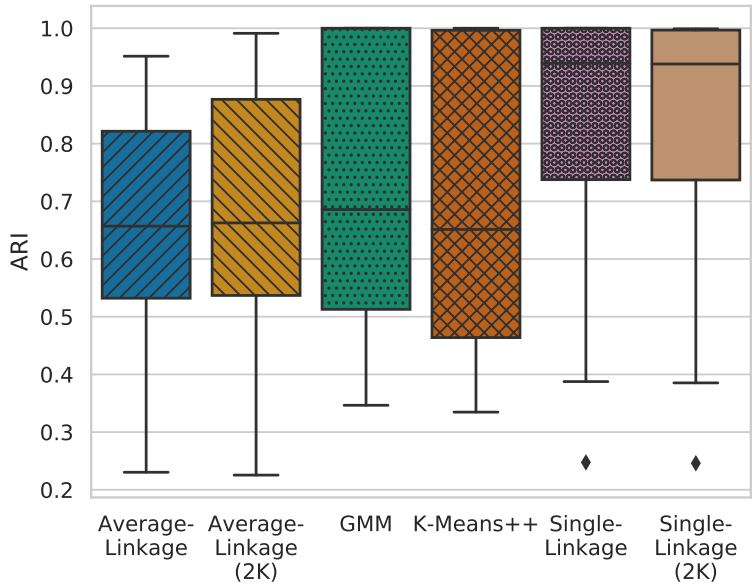


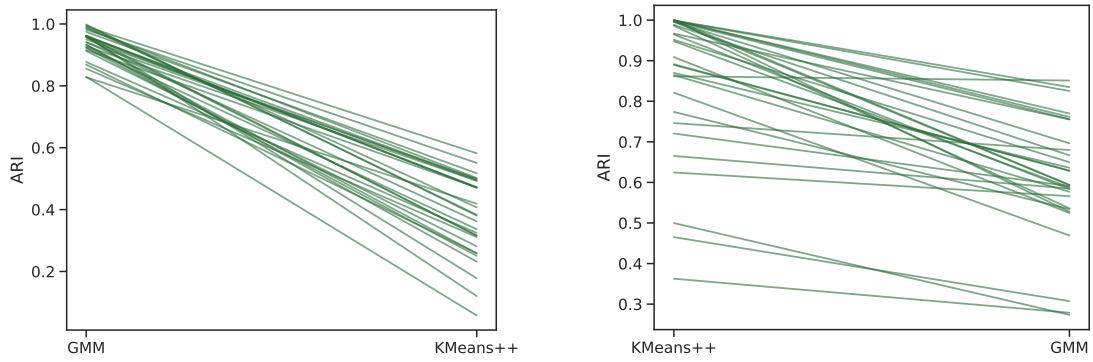
Figure 6.24: Clustering performance across all algorithms for single-linkage vs. average-linkage.

section they are put against each other to study what structures cause performance differences.

GMM and K-Means++

The final combination to study more in-depth is between GMM and K-Means++. As discussed in Section 2.4, K-Means can be formulated as a special case of GMM, and thus broadly similar performance is expected. It is expected, however, that some cluster structures (such as neighbouring clusters that are highly eccentric and orthogonal) are likely to pose more of a problem for K-Means++ than GMM. Owing to its consistent high performance and suitability for the cluster representation of HAWKS, the generation of datasets that are easier for K-Means++ over GMM may be difficult.

The spread of ARIs obtained for both scenarios is shown in Figure 6.25. From the consistency of the results, it is clear that the optimization of GMM vs. K-Means++ was easier than vice versa — the average performance difference is much greater, and the spread of performance for the winning algorithm is much wider for K-Means++. The results indicate that K-Means++ shares more mechanistic similarities with GMM than vice versa, as there is a higher correlation of performance (for both lower and higher ARI values).



(a) GMM vs. K-Means++ (b) K-Means++ vs. GMM

Figure 6.25: The ARIs obtained across the 30 runs for each of the head-to-heads between GMM & K-Means++.

Two example datasets of GMM vs. K-Means++ are shown in Figure 6.26. The first example (Figure 6.26a) is somewhat expected as GMM can handle eccentric clusters, but nonetheless it is encouraging that such an exploitation was found. The compactness-based nature of K-Means++ presents difficulties when clustering neighbouring, highly eccentric clusters (note that, despite the look of the clusters they are separated as $overlap = 0.006$). The second example (Figure 6.26b) is interesting, as it highlights a potential benefit of using metaheuristics. The placement of highly eccentric clusters on top of other clusters is undesirable and unrealistic for a dataset, but highlights the very different nature of the algorithms: the iterative nature of K-Means++ does not permit membership assignment akin to overlapping mixture models. The amount of *overlap* (0.034) is low for this example, and although many algorithms would struggle with this example the relative proportion of overlapping data points indicates that the very low ARI obtained is caused by the placement completely disrupting the convergence of K-Means++.

For K-Means++ vs. GMM, we can see two example datasets produced in Figure 6.27. Similarly to when GMM was the 'losing' algorithm against single-linkage, the initialization is the weak point of the approach that HAWKS exploits. For single-linkage, this exploit utilized elongated clusters (Figure 6.19a), though this cannot work here since that is also a weakness of K-Means++. The improved initialization scheme used by K-Means++ ensures a wider spread of the initial centroids (Arthur and Vassilvitskii 2007), making it less susceptible to large clusters placed far away from other clusters (where it can be erroneously split by trapping

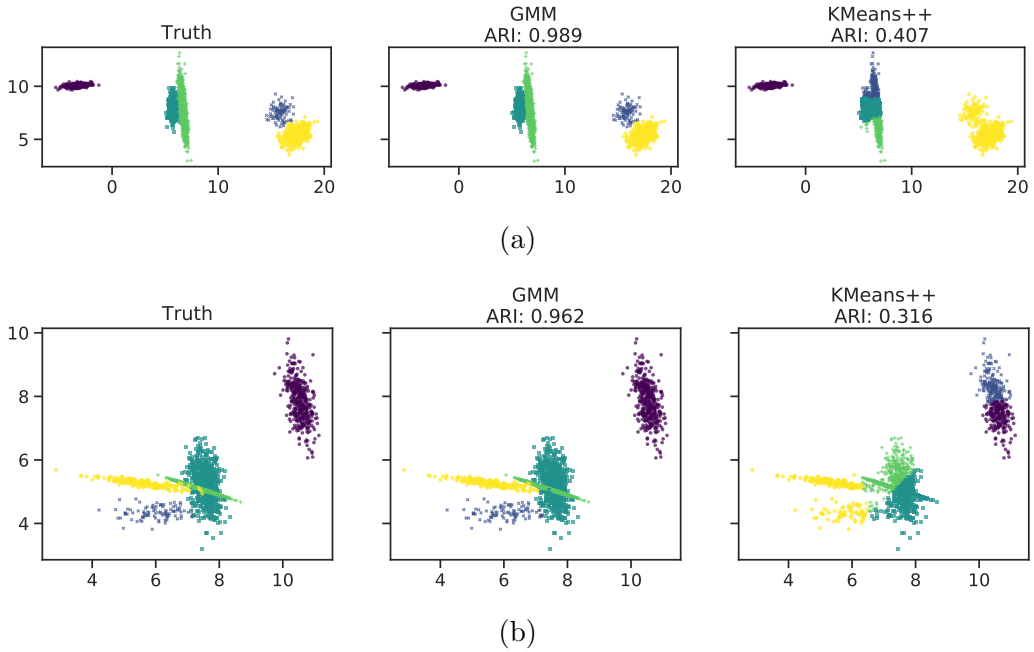


Figure 6.26: Two example datasets from GMM vs. K-Means++, highlighting the use of eccentricity and/or clusters placed on top of each other which mechanistically K-Means++ cannot separate, in contrast to GMM.

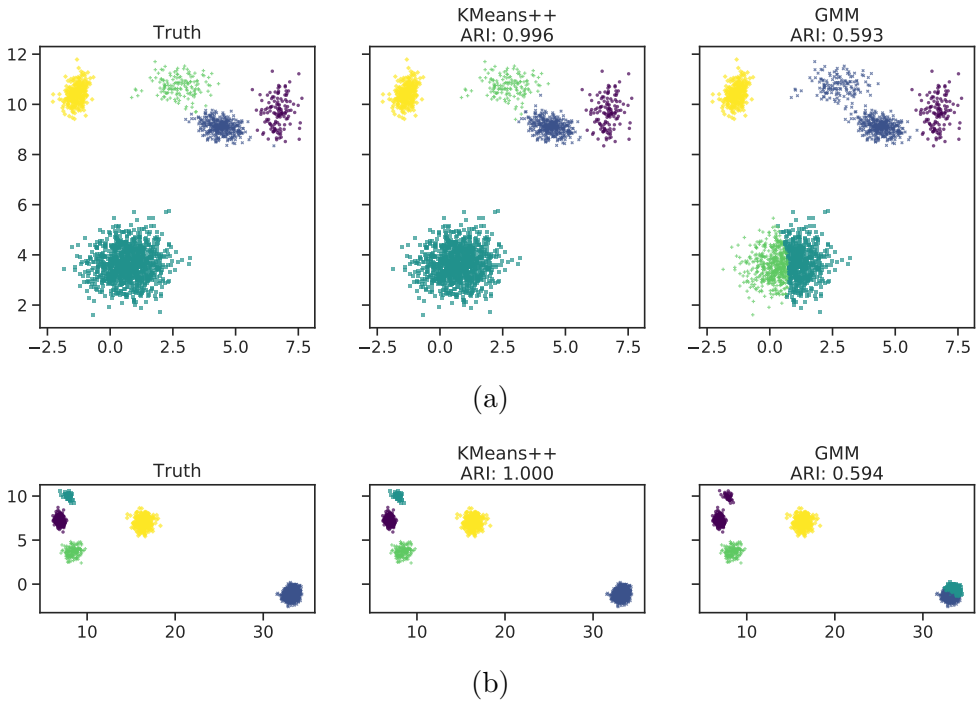


Figure 6.27: Two example datasets from K-Means++ vs. GMM, highlighting a similar exploitation previously used against GMM to place larger clusters further away to induce poor initialization.

two or more of the initialization locations). Lacking this more advanced initialization scheme, GMM more easily converges to the local maxima, the chance of which is increased by HAWKS placing large clusters further away. Experimenting with these algorithms and/or initialization schemes within HAWKS presents a potentially useful avenue, as visualizing the locations of clusters may help generate insights into the pitfalls of the scheme, namely the location of the initial centroids (for K-Means++) or parameter estimates (for GMM).

6.3.3 Experimental summary

In this section, we reformulated the objective function of HAWKS to use the actual cluster assignment of two algorithms, seeking to maximize the difference in their performance (according to the ARI). Rather than tweaking parameters trying to coax HAWKS into generating structures that one algorithm is better suited to when compared to another algorithm, by directly maximizing the performance difference such structures can be discovered. An advantage of HAWKS that was not explored here is the option of iteratively adjusting the parameters based on the datasets produced, such as discouraging eccentricity in GMM vs. K-Means++ to try and discover a less obvious weakness.

By using algorithms that are well-understood, we have been better able to understand whether the structure of the datasets that have been used to generate differences between algorithms are both valid and interesting, as opposed to just overlapping clusters such that the data itself becomes unrealistic. This understanding better facilitates future application to more complex algorithms that are less-understood.

6.4 Summary

This chapter saw multiple improvements to HAWKS, namely: a new mutation operator that better allows for optimizing datasets in higher dimensions (Section 6.1); a clearer understanding of the parameters available in HAWKS and how these can be used to generate more diverse datasets, as measured by a more informative set of problem features (Section 6.2); and, a new mode where we generate datasets specifically to maximize the difference in performance between two algorithms (Section 6.3).

Chapter 7

Adaptive Evolutionary Clustering

In this chapter we look at another use of EAs within clustering, where instead of *generating* clusters we use EAs as a clustering function to directly *assign cluster membership*. In Section 7.1 we provide background on this field of evolutionary clustering by detailing an algorithm (Δ-MOCK) that we then extend with a more flexible encoding in Section 7.2 to create our "Adaptive-MOCK" variant. Then, in Section 7.3 we use our generator (HAWKS) and the instance space/problem feature set we developed in Section 6.2 to further investigate the different methods we use in "Adaptive-MOCK". Some of the content in this chapter has been taken from Shand et al. (2018).

7.1 Background: Evolutionary clustering

Evolutionary clustering is the field where an EA is used to identify cluster membership, given a dataset (or a dissimilarity matrix of data). As discussed in Chapter 2, there is no single measure that can capture the many facets that define a cluster. Naturally, this sets up clustering as a multi-objective optimization problem, for which EAs can be used to optimize clusterings. For more information about the field as a whole, see Handl and Knowles (2004), Handl and Knowles (2007), Mukhopadhyay et al. (2014), and Mukhopadhyay, Maulik, and Bandyopadhyay (2015).

7.1.1 The Δ-MOCK Algorithm

Garza-Fabre, Handl, and Knowles (2017) introduced Δ-MOCK, as an improvement upon the MOCK algorithm (Handl and Knowles 2007). Although there lacks comprehensive work comparing evolutionary clustering algorithms, Δ-MOCK is a state-of-the-art approach that is able to cluster difficult datasets (García and Gómez-Flores 2016; Zhu, Xu, and Goodman 2020).

In this section, we outline how the Δ-MOCK algorithm works, to provide context for the rest of this chapter where we build upon this algorithm. For a complete description of the main changes and differences with the original MOCK algorithm, see Garza-Fabre, Handl, and Knowles (2017).

Representation

Δ-MOCK uses an adaptation of the locus-based adjacency representation, illustrated in Figure 7.1, which was used in MOCK and proposed by Park and Song (1998). This representation treats each of the N data points as nodes on a graph, where the edges on that graph are encoded in the genotype by the index and the value of each gene. For example, in Figure 7.1 the first gene has the value four, thus creating an edge between nodes (synonymous here with data points) 1 and 4 (i.e. $\mathbf{x}_1 \rightarrow \mathbf{x}_4$).[1] The connected components (a subset of nodes connected to each other) of the resulting graph are the clusters, exemplified by the four clusters in Figure 7.1.

The advantages of this representation are: no K is encoded into the representation, allowing for its discovery; there is no bias towards a cluster representation (such as other representations that encode real-valued centroids directly [Mukhopadhyay, Bandyopadhyay, and Maulik 2006]); and, meaningful genetic operators can be created to modify the edges on the graph. A primary disadvantage, however, is that the genotype length is equal to N, thus growing with the size of the data and increasing the search space. If each node can connect to itself or any other node, then the total search space is N^N, which quickly becomes infeasible to search over. As we will see, however, MOCK and Δ-MOCK employ strategies to limit this search space.

The main improvement in Δ-MOCK was the identification that not all edges are equally important, thus enabling the creation of a reduced version of this

[1] The direction is illustrative only as the underlying graph is undirected.

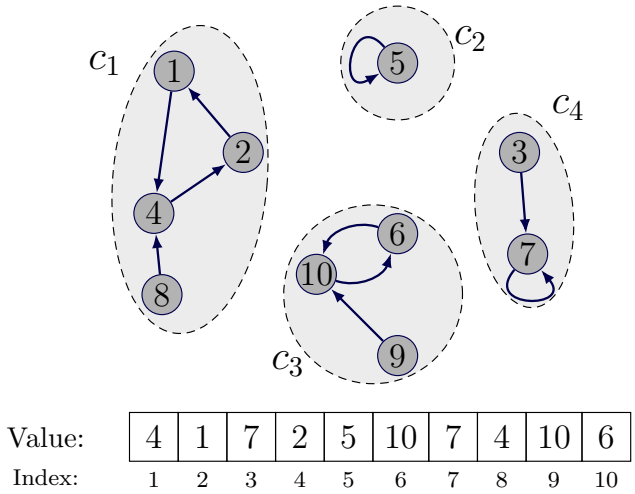


Figure 7.1: Locus-based adjacency representation, where the genotype (of length N) encodes the edges that connect the N nodes (which represent the data points). For the gene at index i with value j, an edge is added to the graph between those nodes, i.e. $\mathbf{x}_i \to \mathbf{x}_j$. The connected components of the graph represent the clusters.

representation. Owing to its namesake, a hyperparameter δ is introduced that represents the percentage of the genotype to *freeze*, i.e. to not involve in the optimization. This then leads to another core piece of Δ-MOCK — the minimum spanning tree (MST). Previously mentioned in Section 2.5.2, the MST is the subset of edges that connect all nodes (without cycles) such that the sum of the weights (which is the distance between points in this case) is minimized. As part of the precomputation, the MST is calculated and used as a starting point for solutions. To identify the relative importance of edges on the MST, the *degree of interestingness* (DI) is used as a measure to identify which edges are likely to be most important during the optimization. This measure was previously used to guide the generation of the initial population in MOCK (Handl and Knowles 2007), but is now utilized to reduce *all* genotypes throughout the optimization.

To define the DI, we need to define notation for the *rank* of a data point with respect to another (in terms of nearest neighbours). In Section 5.1.3 we defined $n_{\mathbf{x}}(j)$ as the jth nearest neighbour of data point $\mathbf{x}$. The inverse of this function, $n_{\mathbf{x}}^{-1} : X \to \{1, \ldots, N\}$, provides the rank of a given data point. For example, if data point $\mathbf{x}_5$ is the 3^{rd} nearest neighbour of $\mathbf{x}$, then $n_{\mathbf{x}}(3) = \mathbf{x}_5$ and $n_{\mathbf{x}}^{-1}(\mathbf{x}_5) = 3$.

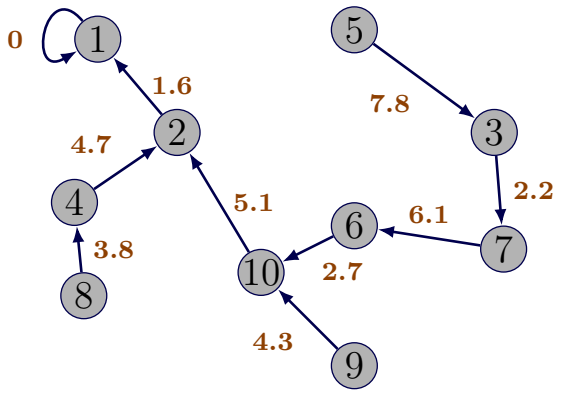
Using this, the DI for an edge $\mathbf{x}_i \to \mathbf{x}_j$ as follows:

$$\mathrm{DI}(\mathbf{x}_i \to \mathbf{x}_j) = \min\left\{n_{\mathbf{x}_i}^{-1}(\mathbf{x}_j), n_{\mathbf{x}_j}^{-1}(\mathbf{x}_i)\right\} + \frac{d(\mathbf{x}_i, \mathbf{x}_j)}{d_{\max}}, \tag{7.1}$$

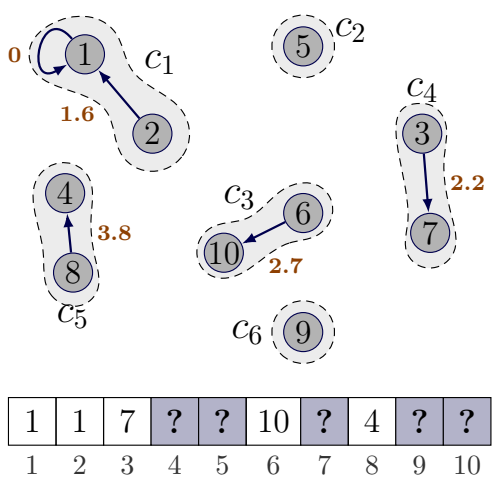
where $d_{\max}$ is the maximum distance between any two data points in the dataset to ensure that the second term is in the range $[0, 1]$. This provides nuance and discrimination between edges of the same nearest neighbour ranking. A higher DI value indicates that the two points are a more unlikely edge in the MST as they are further outside each others local neighbourhood, and is thus potentially an edge between clusters. Note that Δ-MOCK by default uses the Euclidean distance, but other distance measures can be used to calculate the DI and MST.

A ranking is created for all edges in the MST according to their DI value. The hyperparameter δ is then used to partition the edges of the MST into two sets: "relevant" (Γ) and "nonrelevant" (Δ). The parameter δ is a value in the range $[0, 1]$ that specifies the fraction of the least interesting edges (i.e. lower DI value) that are deemed "nonrelevant" and thus frozen for all individuals, effectively reducing the genotype (and therefore search space) to the "relevant" set of edges. When $\delta = 0$, there is no reduction in the genotype length, and thus $|\Gamma| = N$. The reduced genotype length is calculated as $|\Gamma| = \lceil N \times (1 - \delta) \rceil$.

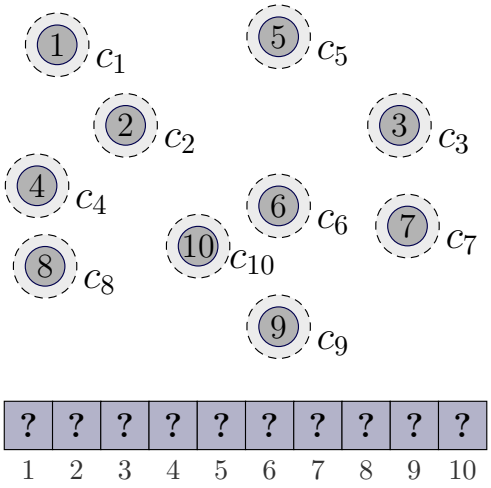
Figure 7.2a shows the MST for our example dataset, alongside a DI value for each edge. If we wish to reduce the search space by half then we can set $\delta = 0.5$, which freezes half of the edges as seen in Figure 7.2b. The reduced encoding allows exploration of only the top 50% most interesting edges (highlighted by "**?**" in the genotype), thereby creating a partial solution with $P = 6$ *components* and a genotype of length $|\Gamma| = 5$. These *components* or sub-clusters consist of points that are likely to belong to the same cluster, transforming the optimization problem so that it is now finding the optimal configuration of these components rather than single data points (though as shown components can contain any number of data points). When $\delta = 0$, the full genotype is used as with the original MOCK (Figure 7.2c) such that each data point is a component ($P = N$). We refer to this as the maximum resolution of the search, in contrast to the minimum resolution ($\delta = 1$) where the MST represents a single-cluster solution. As the value of δ fixes a percentage of edges in the MST, it is clear that the optimal value of δ will differ for different datasets.



(a) MST (with DI values)



(b) Components when $\delta = 0.5$, showing some frozen edges



(c) Components when $\delta = 0$, with no reduction in search space

Figure 7.2: An example dataset with $N = 10$ data points is presented. In (a), the MST is shown (equivalent to $\delta = 1$). The DI value of each edge is displayed next to it. In (b), the $P = 6$ components corresponding to $\delta = 0.5$ can be seen, where half of the least interesting edges have been fixed. This produces a reduced genotype of length $|\Gamma| = 5$. In (c), there are no fixed edges when $\delta = 0$, resulting in a full-length genotype ($P = N$).

Naturally, there is a trade-off between reducing the search space (by increasing δ) and avoiding the creation of components that contain two or more data points that belong to different clusters. Although the non-universally applicable nature of the MST and DI measure means that edges that cross cluster boundaries may not always be found and assigned a high ranking, in general higher δ values increase the size of the components and thus likelihood that they contain heterogeneous data points (with respect to their cluster membership).

Objectives

Δ-MOCK minimizes two objectives: *intra-cluster variance* and *connectivity*. The former objective emphasizes cluster compactness and is minimized when every data point is in its own separate cluster; the latter considers preservation of local structure, and is minimized when all data points are together in a single cluster. This adverse relationship leads to the generation of a Pareto front that spans a range of K values, facilitating both exploration and the natural discovery of appropriate K values.

The intra-cluster variance is simply the sum squared distances for each cluster between each point and its centroid (which is what K-Means sought to minimize in Equation 2.2). We reiterate it here as follows:

$$\frac{1}{N} \sum_{C_k \in \mathcal{C}} \sum_{\mathbf{x} \in C_k} d(\mathbf{x}, \boldsymbol{\mu}_k), \tag{7.2}$$

where $\mathcal{C} = \{C_1, \ldots, C_K\}$ is the set of clusters and $d(\mathbf{x}, \boldsymbol{\mu}_k)$ is the Euclidean distance between point $\mathbf{x}$ and $\boldsymbol{\mu}_k$ (the centroid of cluster k). The Euclidean distance is used by default in Δ-MOCK, but for other inputs other distance measures may be more appropriate.

The second objective, connectivity, was used as a problem feature and defined in Equation 6.5 (except here there is no $1/N$ averaging term). In brief, it measures the extent to which neighbouring data points are assigned to *different* clusters, with larger penalties incurred when the data points have a higher nearest neighbour ranking (i.e. are closer). Δ-MOCK defines the size of the neighbourhood i.e. the number of nearest neighbours to consider by the hyperparameter L.

Underlying Δ-MOCK is the NSGA-II algorithm (a change from MOCK, which used PESA-II [Corne et al. 2001; Garza-Fabre, Handl, and Knowles 2017; Handl and Knowles 2007]), owing to the more elitist nature of the latter imposing too

much selection pressure such that genetic material from the initialization (discussed in the following section) was removed prematurely. The crowding distance used by NSGA-II (Deb et al. 2002) to promote diversity across the Pareto front is additionally useful for Δ-MOCK, as this front contains individuals across a range of K values.

Initialization

As previously mentioned, the initial population is generated using the MST in order to closely approximate the Pareto front. Individuals are created by cutting a random number of edges from the MST, thus also creating individuals of a varied number of clusters. An initial range of k values in the range $[2, k_{\max}]$ is used, where $k_{\max}$ is a user-defined parameter which is twice the real (or estimated if unknown) number of clusters. The upper bound, $k_{\max}$, is overestimated to ensure a variety of different individuals are created, thereby facilitating exploration. Each k is iteratively sampled from this range (without replacement) to generate solutions, which are obtained by removing the $k-1$ most interesting edges in the MST. Each edge $\mathbf{x}_i \to \mathbf{x}_j$ is then replaced by the edge $\mathbf{x}_i \to \mathbf{x}_h$ where $\mathbf{x}_h \in \{n_{\mathbf{x}_i}(1), \ldots, n_{\mathbf{x}_i}(L+1)\} \setminus \{\mathbf{x}_j\}$ i.e. the new edge must be different from the previous one, but is either self-connecting or to one of $\mathbf{x}_i$'s L nearest neighbours. Thus, variation is introduced into the initial solutions beyond the genetic material present in the MST.

Genetic Operators

Standard uniform crossover (Section 3.2.2) is used for recombination. For mutation, the neighbourhood-biased mutation operator is used (Handl and Knowles 2007). The mutation probability is calculated individually for every gene, where for an edge $\mathbf{x}_i \to \mathbf{x}_j$ encoded by a gene the probability is:

$$p_m(\mathbf{x}_i \to \mathbf{x}_j) = \frac{1}{|\Gamma|} + \left(\frac{n^{-1}_{\mathbf{x}_i}(\mathbf{x}_j)}{|\Gamma|} \right)^2 . \tag{7.3}$$

where the second term increases the probability of mutating edges between data points that are further apart (at least in terms of nearest neighbour ranking). When selected for mutation, the edge $\mathbf{x}_i \to \mathbf{x}_j$ is replaced with the edge $\mathbf{x}_i \to \mathbf{x}_h$ in the same way as the initialization. Thus, the mutation can only create edges

to $L + 1$ nodes (the node itself and the L nearest neighbours). While this has the advantage of not creating an edge between two data points that are very far away when $L << N$, if L is too small then there is the possibility that all possible mutations for a given gene have no effect on the fitness (as every edge is between two points in the same cluster). The genetic operators are applied to the reduced genotype only. The design of Δ-MOCK is such that the relevant set (Γ) of edges that we do mutate *should* be on cluster boundaries such that the neighbourhood defined by L is of interest.

7.1.2 The need for adaptation

As stated in the previous section, the δ hyperparameter limits the search space to the (hopefully) more 'interesting' region, providing the opportunity to find equivalent or better solutions at a lower computational cost. As previously mentioned, the flexibility of the locus-based adjacency representation is such that, with no restrictions, the size of the search space is N^N and thus infeasible to search over. The restriction of the L neighbourhood hyperparameter we previously discussed restricts the search space to $(L+2)^N$ instead (the L nearest neighbours, the node itself, and the node in the MST which may have a nearest neighbour ranking higher than L), where typically $L << N$. With the restriction of the search to the relevant set (Γ) to create a reduced genotype of length $|\Gamma|$, the search space for Δ-MOCK is $(L + 2)^{|\Gamma|}$.

The optimal δ value is the highest possible value that does not freeze an edge that connects two clusters, which inherently differs between datasets. As this represents the smallest search space where the optimal solution is reachable, values of δ above and below this value represent a trade-off between performance and computation time. For the more exploratory approach of clustering, a more conservative δ value may be appropriate to provide greater diversity in the final population. As one of the primary improvements of Δ-MOCK was the decreased computation time, the prospect of multiple runs to find this optimal δ seems counterintuitive. Garza-Fabre, Handl, and Knowles (2017) compromised by using a heuristic that scales δ with the size of the dataset. For this the authors used the notation $sr\eta$, which specifies that $|\Gamma| = \eta\sqrt{N}$, where η is a user-defined value. The actual δ is then calculated as $\delta = 1 - \frac{\eta}{\sqrt{N}}$. While this allows for a more universal approach to setting δ, it still may require further tuning.

To avoid repeated runs of Δ-MOCK to find better δ values, in the next section

we propose a novel method of adjusting this parameter *during* the optimization, based on the current rate of convergence, such that the search space is enlarged only when needed.

7.2 Towards an adaptive encoding

This section outlines the method proposed in Shand et al. (2018) to create a variant of Δ-MOCK, "Adaptive-MOCK", that adjusts the search space during the run in order to limit computation and focus the optimization on the edges more likely to produce better clusterings.

7.2.1 Performance-based adaptation

To identify when the search space should be modified, we need to measure the current state of the optimization to understand when we have converged or exhausted the current search space (i.e. explored the relevant combinations of components). Referring to the categorization of control strategies proposed in Eiben, Hinterding, and Michalewicz (1999), the most appropriate description of our proposed method is *adaptive parameter control*. Such methods generally refer to a specific parameter of the EA itself (such as the population size), whereas here we are tuning a parameter that controls the genotype length (by freezing genes to the value of the MST) and thus the search space itself. As such, this deviates slightly from the original definition. For further background, see Section 3.4.

Two similar methods to our proposal are the ARGOT strategy (Shaefer 1987) and delta coding (Whitley, Mathias, and Fitzhorn 1991). The former uses a flexible mapping of function variables to the genes that allow for both the expansion and reduction of the search space through adaptation of the number of bits that are used to represent the gene. By increasing or decreasing the numerical precision, the resolution of the search is modified. Delta coding modifies the search space through iterative reinitializations in a range around the current best solution. The trigger, or condition, used to reinitialize is a measurement of the Hamming distance between the best and worst individuals in the population, and is thus a proxy measure for the diversity (or lack thereof) of the current population. Although the Hamming distance threshold used is fixed, the size of the "hypercube" (i.e. search space) varies dependent on performance.

Algorithm 7.1: Adaptive-MOCK

input : X, δ, $G_{\max}$, L, $k_{\max}$
output: Population of partitions ($\mathcal{P}$)

```
precomputation()                          // MST, DI, "delta-evaluation" etc.
P ← initialization()
evaluation(P)
HV_0 ← hypervolume()
for gen ← 1 to G_max do
    P' ← parental_selection(P)
    P'' ← genetic_operators(P')
    evaluation(P'')
    P ← environmental_selection(P ∪ P'')
    HV_gen ← hypervolume()
    if trigger_mechanism(HV_gen) then
        Reduce δ, recalculate components
        search_strategy()
end
P → Pareto_nondominated()
```

Typically, adaptive parameter control methods rely on feedback from the current search to trigger changes of the parameter values (Karafotias, Hoogendoorn, and Eiben 2015). Measures used to quantify the state of the search generally consider either the quality (i.e. fitness) of the best solution in the current population or the quality of the population as a whole. Of course, with multiple objectives quantifying the population quality is an active area of research (Li and Zheng 2009; Riquelme, Lücken, and Barán 2015; Zitzler and Thiele 1998). In our case, we use the hypervolume to provide feedback on the performance due to its simplicity, popularity, and Pareto-compliant (i.e. adheres to Pareto dominance, defined in Section 3.1.1) property. The hypervolume (discussed in Section 3.2.3) is a unary measure of the volume of the Pareto front dominated by the current population (Zitzler and Thiele 1998). It has been previously used as a convergence measure in the context of a stopping criterion (Guerrero et al. 2009; Trautmann et al. 2008) or to increase the population size (Martínez, Oropeza, and Coello 2011), though here we use it to adapt the search space.

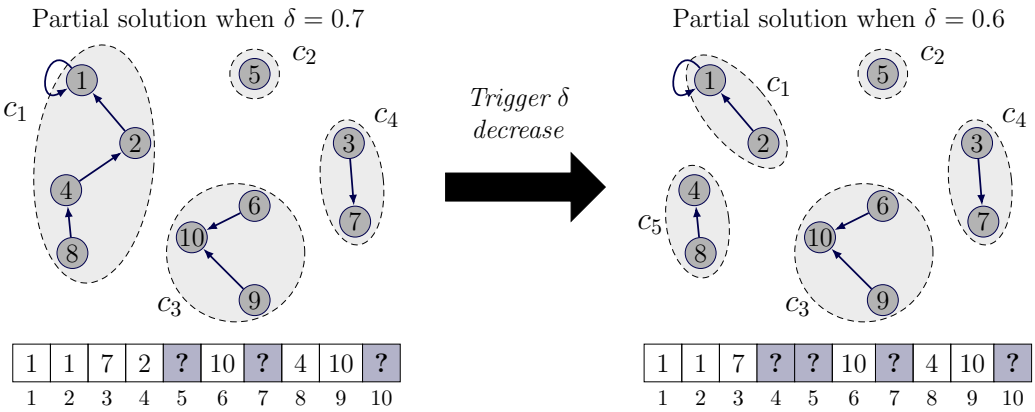


Figure 7.3: Illustration of an expansion of the reduced genotype (signified by the genes with **?**) when δ is decreased. The edge with the next highest DI value in the MST is unfrozen, which removes the edge $4 \rightarrow 2$ and splits component c_1.

7.2.2 Adapting Δ-MOCK

Algorithm 7.1 provides pseudocode for Adaptive-MOCK, where the primary differences from Δ-MOCK are in lines 11–13. In this and the following sections, we motivate and detail the additions we made in Adaptive-MOCK.

Our adaptation of the search space occurs through incremental changes to the δ value. A stagnation of the hypervolume indicates that better (Pareto-dominant) solutions are not being found, which indicates that the search space is too restricted. In order to focus the search and limit computation as much as possible, we start with a restricted search space (i.e. a very high δ value), and incrementally decrease this value when the hypervolume stagnates. Specifically, we consider the situation with a given set of T levels of resolution, with the minimum resolution level (i.e. smallest search space) defined by δ_{High} and the maximum resolution defined by δ_{Low}.

To understand what changes occur with a decrease in δ, Figure 7.3 illustrates this on our example dataset. The next most 'interesting' edge (which is $4 \rightarrow 2$ according to the DI values shown in Figure 7.2a) is removed, splitting apart component c_1 into two disjoint components. As a result, data points 4 & 8 can form clusters without the presence of data points 1 & 2 (and vice versa), which may present a better partitioning.

A stated advantage in the design of Δ-MOCK was the "delta-evaluation", which allowed for the precomputation of a significant part of the fitness evaluation (Garza-Fabre, Handl, and Knowles 2017). By computing the objective

contributions of the components, the objective function for a given individual requires identifying which components are connected and then computing the contributions of these combinations. Further details for this approach can be found in Garza-Fabre, Handl, and Knowles (2017). Of importance to us is the need to re-do this precomputation when δ changes, which needs to be taken into account when comparing computation time between Δ-MOCK and Adaptive-MOCK.

There are two main components to our method that we explore: the *trigger mechanism* that causes a change in the resolution, and a *search strategy* that helps to search the newly expanded search space. These are explained in Section 7.2.3 and Section 7.2.4 respectively.

7.2.3 Trigger mechanisms

To determine whether the hypervolume-based mechanism adds additional value over a non-performance-based trigger, we compare against two baseline trigger mechanisms. The three mechanisms we experiment with are as follows:

1. *Random* — Randomly sample $T-1$ generations at which to trigger a decrease in δ.

2. *Interval* — Create $T-1$ equally spaced intervals to provide an equal number of generations at each δ value.

3. *HV* — Use the rate of change in the hypervolume to determine whether the search is stagnating and thus whether a decrease in δ is required.

To explore T levels of resolution, we have a maximum number of $T-1$ triggers. The two baseline trigger mechanisms are prescriptive in that there is no variability in the number of triggers. Therefore, for a fair comparison the HV method cannot trigger more than $T-1$ times (though, depending on performance, it may trigger less than this). The *random* mechanism simply draws $T-1$ samples from the range $[1, \ldots, G_{\max}]$. The *interval* mechanism creates T equal intervals between 0 and $G_{\max}$ to give an equal number of evaluations at each level of resolution. For example, if $G_{\max} = 100$, each level of resolution is explored for 20 generations.

For our hypervolume-based approach, in order to identify convergence we require a reference to compare against. For this, we calculate the change in hypervolume (ΔHV) in the first 10% of generations ($G_{\max}/10$). After this, a moving average gradient of the hypervolume is recorded, and when this average

falls significantly below the reference gradient ($< 0.1 \times \Delta HV$), this indicates that the rate of improvement has notably stagnated and thus we should expand the search space (i.e. *trigger* a decrease in δ). To avoid multiple, rapid changes in δ, after a trigger we record a new reference gradient for the following $G_{\max}/10$ generations (during which δ cannot be decreased again).

7.2.4 Search strategies

As a change in resolution results in an expansion of the current search space, it follows that it would be beneficial to focus on exploring this new space specifically, avoiding repeated evaluations on previously explored component combinations. For this, five strategies are considered:

1. *CO* — 'Carry on', a baseline strategy with no changes following a decrease in δ.

2. *FM* — Fair mutation, where new individuals are created specifically to explore each of the newly unfrozen genes.

3. *RO* — Reinitialized offspring, where offspring are generated using the specialized initialization scheme.

4. TH_{all} — Triggered hypermutation applied to all genes in the (reduced) genotype.

5. TH_{new} — Triggered hypermutation applied to only the newly unfrozen genes of the (reduced) genotype.

Fair mutation (*FM*) was a method introduced in Allmendinger and Knowles (2010) in order to explore the space of solutions from newly introduced genes for a binary genotype. Here, we modify this method for Δ-MOCK's representation. Offspring are created by generating an equal number of individuals for each new gene in the genotype. In the original work, each new gene was set to 1 and protected from modification for several generations in order to explore genotypes in the context of the new gene, assisted through the use of a raised mutation rate for the remaining genes. As every individual in our population will have the same value for the newly unfrozen genes (the value present in the MST), individuals are generated separately for each new gene by setting this value as self-connecting, thus removing the edge. Using Figure 7.3 as an example, as there is only a single

gene (at index 4) added to the genotype all offspring would have a value of 4 here protected for several generations. This helps to explore previously unseen partitions as the new component(s) are separated (at least from that node). This shares similarities with the binary genotype variant of Δ-MOCK introduced in Garza-Fabre, Handl, and Knowles (2017), which just encodes whether the edge specified by the MST is present or not, directly searching for the optimal combination of MST-derived components.

Owing to Δ-MOCK's specialized initialization routine, which as previously discussed is bound by the relevant set (Γ), the *RO* strategy uses this routine to generate offspring to compete against the current population. As the purpose of this routine is to be an approximation of the Pareto front, this should produce competitive offspring and, with the random mutation used in this routine, also introduce useful *new* genetic material into the population.

As discussed in Section 3.4.1, hypermutation is an approach where mutation rates are *dramatically* (i.e. ×500) increased for a single generation. In the original work (Cobb 1990), this hypermutation is employed following identification of a drop in the average fitness. Here, we use it as a strategy to rapidly explore the search space to find better solutions, with the addition that in our work this is accompanied by a change in the search space itself. The first strategy we use based on this concept, TH_{all}, applies the hypermutation rate to all genes in the (reduced) genotype, whereas the second strategy (TH_{new}) applies this *only* to the genes that have become available as a result of the change in δ. In Figure 7.3, TH_{all} would apply this rate to all 5 of the genes in the reduced genotype, whereas TH_{new} would apply this rate to the gene at index 4 only. As such, the second is more focused on specifically exploring potentially different component combinations available via the new genes, whereas the former may be able to do this with the additional context of the other genes. Naturally, the utility of either scheme depends on the structure of the MST and genetic material in the population at the time. Specific parameters for all search strategies are given in the following section.

7.2.5 The effectiveness of adaptation

In this section we examine whether the search strategies with a performance-based trigger mechanism are able to better explore the search space, or at least able to match Δ-MOCK that uses a δ value known to be effective on the datasets

being used (when starting with a δ that is worse on average). As we start the algorithm with a shorter genotype length, this should confer a reduction in computation time. Of course, when the appropriate δ is unknown then this adaptive approach should have additional utility, but here we test that we are able to effectively reduce the δ from a potentially poor value without a considerable loss of performance.

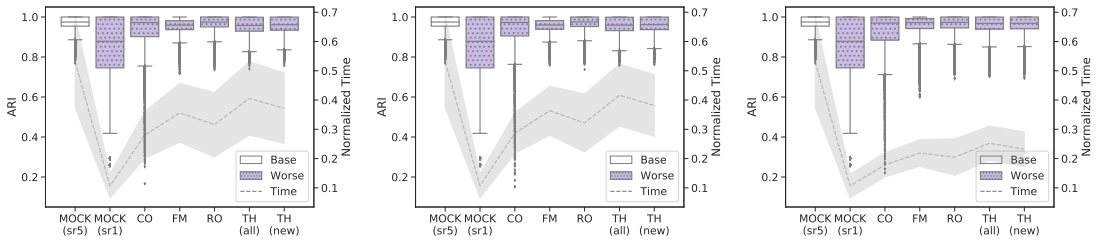
Experimental Design

All five search strategies were run with each of the three trigger mechanisms 30 times across two sets of datasets. We use the 350 (expanded) *HK* and 8 *UKC* real-world datasets from Section 6.2.1.

For each search strategy we begin with a very large reduction in the search space by using $\delta_{\text{High}} = sr1$, i.e. $\delta_{\text{High}} = 1 - \frac{1}{\sqrt{N}}$. This is the first of $T = 5$ resolution levels, down to the largest search space (or lowest level of resolution) which is determined by $\delta_{\text{Low}} = sr5$. We compare this against two baselines: Δ-MOCK (as implemented in Garza-Fabre, Handl, and Knowles [2017]) run at both our highest and lowest level of resolution i.e. $\delta \in \{sr5, sr1\}$. These values were selected as they were found to be on average good and bad (respectively) on the datasets studied here[2]. Assuming that $sr1$ is too restrictive, there should be a trade-off between performance and computation time between these two baseline methods. Our three trigger mechanisms then operate as previously discussed, decreasing δ stepwise after each trigger from $sr1$ to $sr5$, increasing the genotype length by $\sqrt{N}$ each time.

Each run is stopped after 100 generations (i.e. $G_{\max} = 100$). We use a population size of 100, $L = 10$ (the neighbourhood size), and crossover probability $p_r = 1.0$. These parameters are consistent with Garza-Fabre, Handl, and Knowles (2017). The moving average calculated for the hypervolume is a window of 3 generations. For the hypermutation search strategies (TH_{all} and TH_{new}) a $500 \times p_m$ mutation rate is used for a single generation. The *FM* strategy uses a $50 \times p_m$ mutation rate for the non-protected genes, and the designated self-connecting genes are protected for 3 generations.

[2] In Garza-Fabre, Handl, and Knowles (2017), $sr1$ was found to lead to poor performance across these datasets. For some datasets the resulting δ value led to $|\Gamma| < K$, which inherently makes the optimal solution unreachable. As we do not know the optimal δ value, we can only say that *generally* these settings lead to good and poor values of δ.



(a) *Random* trigger (b) *Interval* trigger (c) *HV*-based trigger

Figure 7.4: Performance for each search strategy for each of the three trigger mechanisms for the *HK*-generated datasets. The computation times are normalized in the range $[0, 1]$.

Results

Throughout this section, unless otherwise stated we select the best individual from the final population (according to the ARI) at the end of each of the 30 runs. We select the best solution so that we can assess the potential of each method to generate good solutions. As some Pareto-optimal solutions will inevitably have significantly more or fewer clusters than the true K (leading to poor ARI values), aggregated values across the final population are of less interest here.

In Figure 7.4 we show the clustering performance on the *HK* datasets for all methods, separated by the trigger mechanism used. The computation times are also shown for each search strategy (scaled to the range $[0, 1]$ such that the maximum computation time is set to 1), which includes the additional precomputation required for "delta-evaluation" (discussed in Section 7.2.2). Shading of the boxplots is used to identify whether the result is significantly different in comparison to the baseline method (Δ-MOCK with $sr5$) using a Wilcoxon signed-rank test (with $p = 0.05$ significance level, corrected with the Holm-Bonferroni method [Holm 1979] as we make several pairwise comparisons).

Across all search strategies and trigger mechanisms, none were statistically significantly better than the baseline, Δ-MOCK ($sr5$). Nevertheless, it is clear that the *RO* strategy performed the best out of the search strategies, with results visually similar to the baseline method for a vast reduction in the computation time, with little overhead compared to the baseline method *CO*. In Table 7.1 we can see the mean (and standard deviation) ARI values obtained across the search strategies and trigger mechanisms. From this and the boxplots, we can see that the *RO* strategy performs slightly worse with our *HV*-based trigger mechanism, which is likely due to the more conservative nature of this method resulting in

Table 7.1: ARI mean and standard deviation for *HK* datasets

Method	*Random*	*Interval*	*HV*
CO	0.929 ± 0.106	0.931 ± 0.104	0.921 ± 0.113
FM	0.950 ± 0.046	0.951 ± 0.045	0.951 ± 0.063
RO	0.962 ± 0.048	0.963 ± 0.046	0.956 ± 0.057
TH_{all}	0.950 ± 0.051	0.951 ± 0.050	0.953 ± 0.057
TH_{new}	0.953 ± 0.048	0.954 ± 0.047	0.954 ± 0.056
Δ-MOCK ($sr5$)		0.965 ± 0.042	
Δ-MOCK ($sr1$)		0.846 ± 0.155	

a higher final δ value, thus reducing the number of initializations and therefore impact of this search strategy (which immediately benefits from a change in δ). The significantly poorer performance of Δ-MOCK with the smallest genotype (δ_{High}, or $sr1$) indicates that the starting resolution for the adaptive methods *is* restrictive, and thus performance has been largely recovered when compared to Δ-MOCK ($sr1$). As previously discussed, however, there are many datasets for which this value is not restrictive, leading to a large spread in performance.

Of note is the large number of outliers (and higher standard deviation) found for the *CO* strategy, which indicates that the general idea of using a method to rapidly explore the new space is beneficial and, for some datasets, necessary to recover performance. All of the search strategies studied here resulted in a mean ARI closer to Δ-MOCK ($sr5$) than to the *CO* search strategy, highlighting their utility. When we compare *CO* to the restrictive baseline, Δ-MOCK ($sr1$), for some of these datasets we can see (from the outliers in Figure 7.4) that the expansion of the search space with no additional strategy is actually more detrimental to performance over searching in a restricted space. This is likely due to the reduction of mutation probabilities as the genotype length increases, and the decreasing relevance of the new genes added which reduces the focus of the search.

In Table 7.2, the mean and standard deviation of the normalized computation times are shown. The *HV*-based trigger method greatly reduces the computation required, which as we previously saw corresponds with no considerable drop in performance when compared to the other trigger methods. For the *RO* strategy

Table 7.2: Normalized computation time mean and standard deviation for *HK* datasets

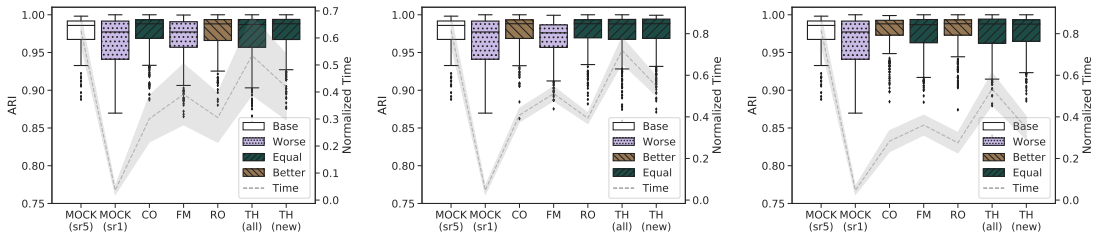
Method	*Random*	*Interval*	*HV*
CO	0.281 ± 0.080	0.287 ± 0.069	0.178 ± 0.041
FM	0.356 ± 0.101	0.364 ± 0.084	0.219 ± 0.046
RO	0.316 ± 0.110	0.322 ± 0.099	0.205 ± 0.063
TH_{all}	0.406 ± 0.125	0.417 ± 0.106	0.252 ± 0.059
TH_{new}	0.372 ± 0.121	0.382 ± 0.106	0.232 ± 0.060
Δ-MOCK ($sr5$)		0.531 ± 0.156	
Δ-MOCK ($sr1$)		0.107 ± 0.041	

with the *HV* trigger mechanism, the mean ARI (0.956) is close to the baseline (0.965) in absolute terms, for a computation time that is (on average) 39.4% that of the same baseline. Such vast differences in computation time become paramount for the practical application of Δ-MOCK on very large datasets. Of note is the higher computation time for the three methods that increase the mutation rate, in particular the two hypermutation strategies, which we discuss further in Section 7.2.6.

Although we did not find that our adaptive approach was able to reach the performance of the baseline method, in Table 7.2 and Figure 7.4 we can see that close performance was achieved for a significant reduction in computation time (even with the repeated precomputation steps). This reduction is most obvious in our *HV*-based trigger mechanism, which illustrated that it was not always necessary to decrease δ down to $sr5$. Across the *HK* datasets, the *HV*-based trigger mechanism decreased δ on average 1.72 times[3].

For the *UKC* datasets, Figure 7.5 shows the ARI and computation times. For these datasets, we can see a mix of statistically equal and even better approaches for some strategies when compared to the Δ-MOCK ($sr5$) baseline. When looking at both these boxplots and the mean ARIs in Table 7.3, we can see that the results are numerically close between all strategies, trigger mechanisms, and even the two baselines. Owing to the large size of the datasets, it is clear that even δ_{High} ($sr1$) is not overly restrictive for this dataset, though the poorer performance of the

[3] A maximum of 4 is possible.



(a) *Random* trigger (b) *Interval* trigger (c) *HV*-based trigger

Figure 7.5: Performance for each search strategy for each of the three trigger mechanisms for the *UKC* datasets. The computation times are normalized in the range $[0, 1]$.

baseline with this δ value does indicate that it was not optimal. The larger size (N) and low cluster count (≈ 11) of these datasets is such that even δ_{High} is not so high that the components are significantly heterogeneous (with regards to the true cluster membership of the points), highlighting a disadvantage to the $\sqrt{N}$-based heuristic of setting δ.

Of interest is the significantly better performance of *CO* for the *interval* and *HV* trigger mechanisms. This is likely due to the non-restrictive nature of the δ values being used, allowing Δ-MOCK to initially focus on the more useful genes in the relevant set (Γ) that exist above δ_{Low}. This highlights that the search strategies which raise the mutation rate perform the worst here, and can be disruptive enough that doing nothing is more effective. Of particular interest is *FM*, which only showed parity with the baseline for the *HV* trigger mechanism and was otherwise worse. This is likely due the protection given to individuals that keep the new components separated, when (as indicated by the high performance of our 'restrictive' δ value) these components are likely homogeneous and thus should not be separated.

The computation times (tabulated in Table 7.4) show a similar pattern to the *HK* datasets, with the *HV* trigger mechanism resulting in a noticeably lower computation time. For the *RO* strategy, we obtain statistically significant results for the ARI with the *HV*-based mechanism, for 33.75% of the (average) computation time. On average, the *HV* trigger mechanism was activated 2.13 times, hence the large reduction in computation time compared to Δ-MOCK ($sr5$). The impact and importance of δ on larger datasets is highlighted by the *far* lower computation time for Δ-MOCK ($sr1$) and the higher variance in computation time for the *random* trigger mechanism.

Table 7.3: ARI mean and standard deviation for *UKC* datasets

Method	*Random*	*Interval*	*HV*
CO	0.975 ± 0.027	0.976 ± 0.025	0.976 ± 0.025
FM	0.966 ± 0.033	0.966 ± 0.029	0.974 ± 0.028
RO	0.976 ± 0.026	0.976 ± 0.026	0.976 ± 0.025
TH_{all}	0.970 ± 0.035	0.973 ± 0.030	0.973 ± 0.032
TH_{new}	0.974 ± 0.030	0.974 ± 0.030	0.975 ± 0.027
Δ-MOCK ($sr5$)		0.975 ± 0.024	
Δ-MOCK ($sr1$)		0.963 ± 0.034	

Table 7.4: Normalized computation time mean and standard deviation for *UKC* datasets

Method	*Random*	*Interval*	*HV*
CO	0.301 ± 0.086	0.314 ± 0.025	0.218 ± 0.039
FM	0.391 ± 0.113	0.393 ± 0.027	0.277 ± 0.037
RO	0.305 ± 0.091	0.302 ± 0.020	0.212 ± 0.038
TH_{all}	0.535 ± 0.146	0.553 ± 0.048	0.410 ± 0.060
TH_{new}	0.418 ± 0.121	0.421 ± 0.036	0.266 ± 0.043
Δ-MOCK ($sr5$)		0.627 ± 0.052	
Δ-MOCK ($sr1$)		0.036 ± 0.016	

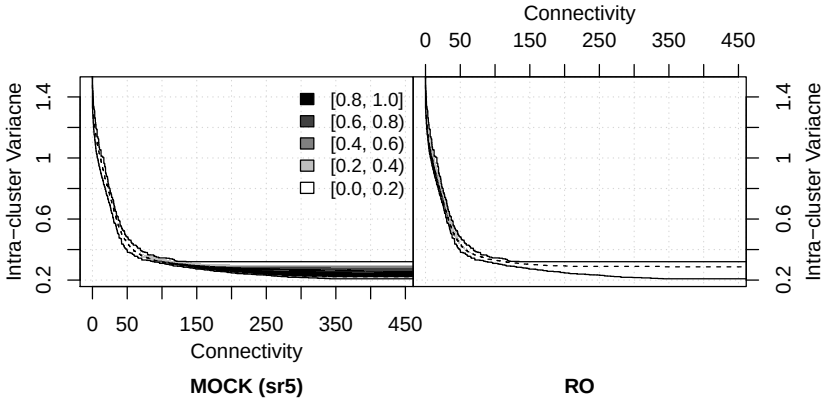
For the *UKC* datasets (and to a lesser extent the *HK* datasets), the computation times for both of the *TH* strategies were noticeably higher than the other methods. Even the *FM* method also had a slightly higher computation time compared to *CO* and *RO*. As the computation time for the raised mutation rate is negligible, there must be a downstream cause of these methods that results in this difference, likely occurring in the evaluation function. In the next section (Section 7.2.6), we investigate this further.

To further explore the differences between the search strategies, we use EAF difference plots (which were introduced in Section 3.2.3). These plots highlight (via shading) the probability of that approach finding solutions in that area of the objective space compared to the other approach depicted. In Figure 7.6 we can see two EAF difference plots comparing the baseline method against *RO* on the same dataset, where Figure 7.6a is when the *HV* trigger mechanism is used and Figure 7.6b shows the *interval* mechanism. In both of these examples, we can see a clear superiority of Δ-MOCK with $\delta = sr5$ when optimizing the intra-cluster variance objective, whereas there is a smaller probability that the *RO* strategy finds better solutions for the connectivity objective. This is likely due to the introduction of genetic material from the MST, which inherently favours solutions that optimize this objective (as the MST itself is the minimization of this objective). We can see that the Pareto front found by the two approaches is more similar with the *interval* mechanism, whereas the higher δ values used for the *HV* mechanism are more likely to favour the connectivity objective (due to a higher chance of a lower number of clusters).

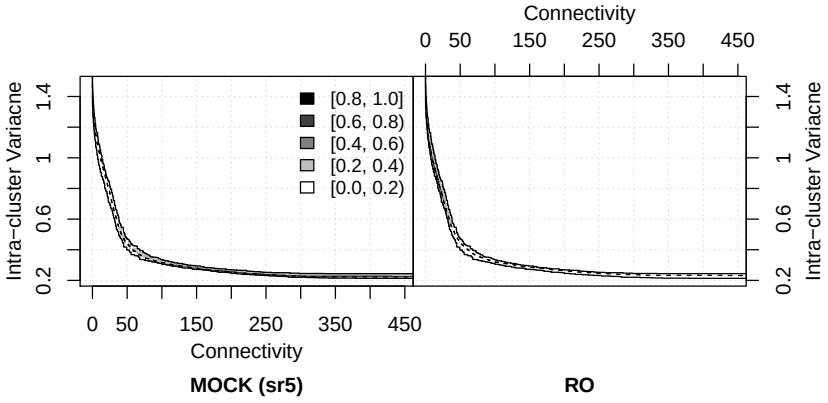
7.2.6 Mutation bias

One stated advantage of MOCK was that, owing to the design of its objectives, the Pareto front produced is comprised of solutions with varying K values. The spread of this front (i.e. the variance of K across the individuals) provides some indication into the diversity of the population, and thus it is useful to look into whether our modifications to the algorithm affected this diversity.

In Figure 7.7 we can observe the spread of the number of clusters found across the final population for one *HK* dataset (a) and one *UKC* dataset (b), where the true K value is shown by the dashed horizontal line on both graphs. Here we can see a clear bias, particularly for the *UKC* dataset, towards a higher number of clusters for all three methods that raise the mutation rate. As an increased

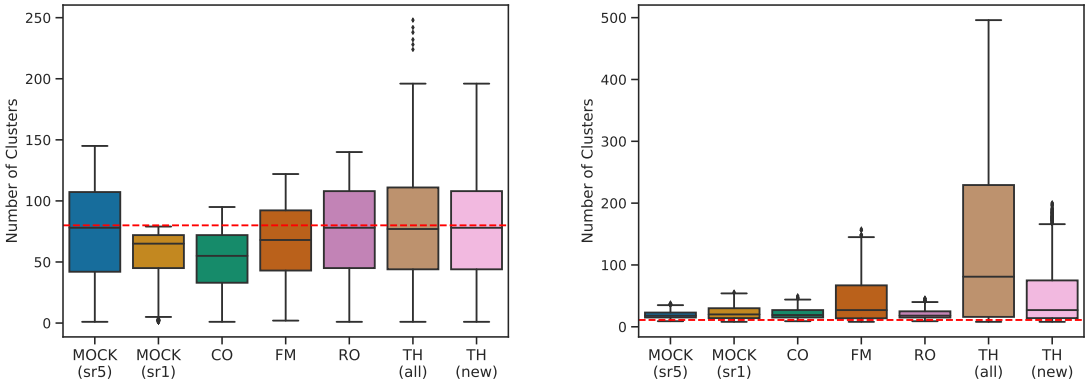


(a) *HV* trigger mechanism



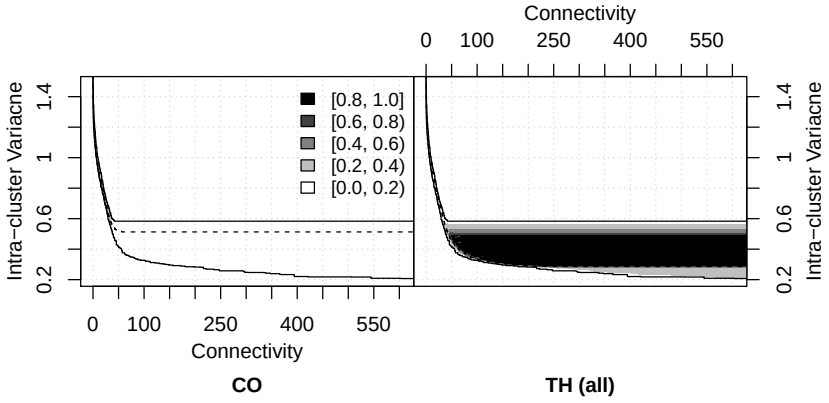
(b) *Interval* trigger mechanism

Figure 7.6: EAF difference plots comparing our baseline strategy (Δ-MOCK with $\delta = sr5$) against *RO* using the *HV* (a) and *interval* (b) trigger mechanisms. Although slight, the *RO* strategy favours the connectivity objective, whereas the original Δ-MOCK is far superior at the intra-cluster variance objective.

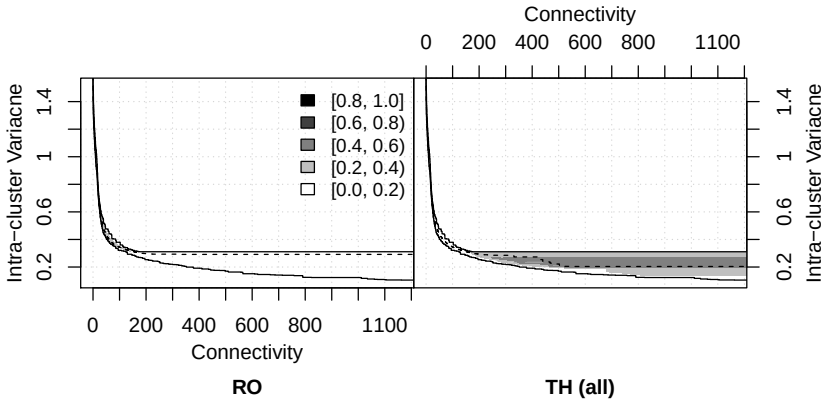


(a) *Random* trigger (b) *Interval* trigger

Figure 7.7: The number of clusters encoded by individuals in the final population for each method on a *HK* dataset (a) and a *UKC* dataset (b). The true number of clusters is shown by the dashed horizontal line.



(a) CO and TH_{all}



(b) RO and TH_{all}

Figure 7.8: EAF difference plots comparing the hypermutation search strategy TH_{all} against CO (a) and RO (b), highlighting the preference towards the intra-cluster variance. Both are runs where the HV trigger mechanism is used.

number of clusters increases the computation required for the fitness evaluation, this is the main source of increase in the computation time required by these methods.

Then, we must further investigate *why* our methods, specifically those that increase the mutation probability, leads to a significant increase in the number of clusters. As previously discussed, the mutation operator works by choosing to replace an edge randomly from a data point's nearest neighbour. This is inherently more likely to create an intra-cluster edge, as nearest neighbours are more likely to be in the same cluster than in a different one. By having a bias towards creating *intra*-cluster rather than *inter*-cluster edges, and as this operator occurs more frequently throughout the genotype, there is a higher chance of increasing

the number of clusters. As this inherently satisfies the intra-cluster variance objective, these solutions are not necessarily rejected, though the crowding distance of NSGA-II should limit the number of solutions in the population that favour this objective, which is evidenced by the wide spread of clusters identified in the final population. Naturally, this spread of focus away from K values closer to the true number reduces the amount of search in the more 'interesting region', which may (in general) affect the ability of these methods to find good solutions, though this is traded for greater diversity. In our experiments, the best individual found (in terms of ARI) did not seem unduly affected by this wider spread of K values, but a slightly lower average ARI was observed.

To further investigate the difference of the hypermutation methods, Figure 7.8 shows the hypermutation method TH_{all} against the baseline search strategy *CO* (Figure 7.8a) and against the best-performing search strategy *RO* (Figure 7.8b). Against both methods, though particularly when compared to *CO*, we can see a greater degree of optimization for the intra-cluster variance when using TH_{all}, further supporting the idea that the mutation operator favours the creation of intra-cluster edges, resulting in higher individuals that encode a higher K and thus favouring the intra-cluster variance objective.

7.2.7 Experimental summary

In this section, we used the *HK* and *UKC* datasets to investigate whether our adaptive method of increasing the search space based on the rate of change of the hypervolume was useful to maintain performance while decreasing the computation time. We also investigated whether our proposed search strategies helped to rapidly explore this new search space. Overall, for these datasets, we found mixed performance. For the *HK* datasets, none of our search strategies was able to achieve significantly better ARI values, though the difference is practically very small. This was, however, obtained for a much lower computation time (the best-performing search strategy, *RO*, required 33.75% of the computation time of Δ-MOCK using $sr5$). For the *UKC* datasets, several of our methods were found to be statistically equivalent or better than the baseline, highlighting the potential of our approach to not only reduce the computation time, but to narrow the search such that performance is improved.

The use of search strategies that increased the mutation rate highlighted a bias in the mutation operator towards one of the objectives, which itself is optimized

as the number of clusters encoded by the individual increases. This was due to the mutation operator being, by definition (for typical L values used), more likely to create edges that are intra-cluster rather than inter-cluster. While we found that this can be useful when the search space is very restricted, when the search space is properly defined this is likely detrimental behaviour.

In the next section, we utilize our work from Chapters 5 and 6 to see if further insights can be gained into these search strategies and the capabilities of MOCK as a whole.

7.3 Can HAWKS grant further insights?

As a further investigation into these methods, we can use the datasets we generated in Chapter 6 to try and provide further information about the differences between these methods. In this section, we take a small selection of these datasets and run both Δ-MOCK and Adaptive-MOCK on them.

We use the 20 datasets produced in Section 6.2.5, where we explicitly attempted to generate more eccentric clusters. These datasets are run using the same settings as were used with the *HK* and *UKC* datasets.

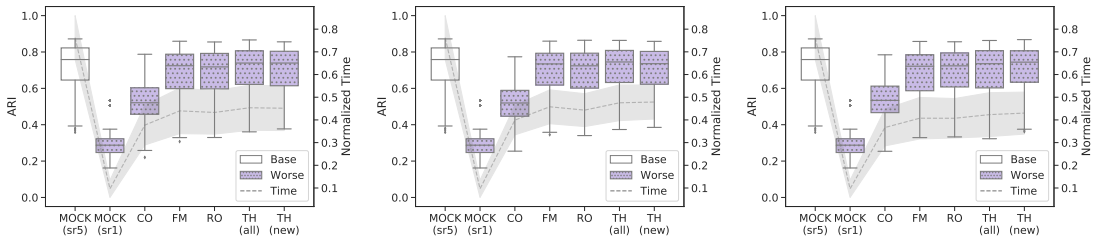
7.3.1 Results: Clustering performance

The performance of the search strategies and the two baseline Δ-MOCK strategies is shown in Figure 7.9. From these boxplots and the mean ARI values (Table 7.5), it is immediately clear that all search strategies were vastly superior to *CO* and that the minimum resolution ($sr1$) was far too restrictive for these datasets.[4] In general, Δ-MOCK and Adaptive-MOCK were able to find good solutions, though with greater variance in performance over the results seen in Figure 6.14b.[5] The skew of the results (with the median being notably higher than the mean), indicates that for some datasets $sr5$ was either too restrictive (or they pose a significantly higher difficulty, which is less likely).

Although none of the search strategies were statistically better than or equal to Δ-MOCK ($sr5$), once again in absolute terms the ARIs were not far apart,

[4] In Section B.1.4 we use a wider range of δ values to determine that $sr5$ was not too restrictive, but the use of $sr9$ resulted in slightly higher ARIs.

[5] Though these clustering algorithms are markedly more simple than Δ-MOCK, for these datasets they were given the true K which is not always known.



(a) *Random* trigger (b) *Interval* trigger (c) *HV*-based trigger

Figure 7.9: Performance for each search strategy for each of the three trigger mechanisms for the HAWKS datasets. The computation times are normalized in the range $[0, 1]$.

Table 7.5: ARI mean and standard deviation for HAWKS datasets

Method	*Random*	*Interval*	*HV*
CO	0.521 ± 0.118	0.515 ± 0.115	0.528 ± 0.115
FM	0.688 ± 0.119	0.696 ± 0.118	0.681 ± 0.126
RO	0.684 ± 0.123	0.686 ± 0.123	0.689 ± 0.122
TH_{all}	0.706 ± 0.114	0.711 ± 0.110	0.701 ± 0.120
TH_{new}	0.702 ± 0.115	0.702 ± 0.113	0.707 ± 0.117
Δ-MOCK ($sr5$)		0.719 ± 0.120	
Δ-MOCK ($sr1$)		0.300 ± 0.090	

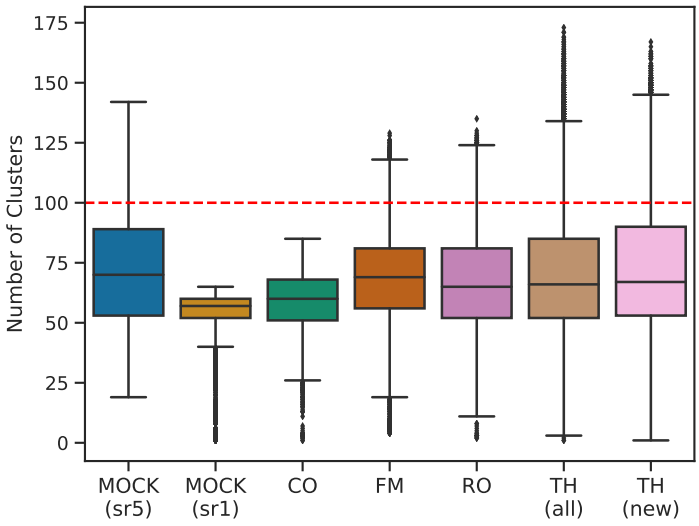


Figure 7.10: The number of clusters encoded by individuals in the final population for each method aggregated across all 20 HAWKS datasets. The true number of clusters (the same for each dataset) is shown by the dashed horizontal line.

particularly for the triggered hypermutation (*TH*) methods, indicating that for this scenario an exploration of a larger number of clusters was beneficial. Figure 7.10 shows the variance in the number of clusters obtained by individuals in the final population (using the *HV* trigger mechanism) across all 20 HAWKS datasets. This further supports that the δ value employed may have been too restrictive and thus explains the utility of the increased mutation rate. The effect of the adaptive approach can be seen with the lower whiskers for the search strategies, as the lower K individuals encoded by the shorter genotypes remain in the population. As the genetic material in MOCK is derived from the MST, all individuals have the same value in the genes that are 'unfrozen' following a decrease in δ, and therefore both the crossover operator and the unmodified mutation operator are unable to significantly modify the genotype to recover the performance (as illustrated by the narrow spread of K for the *CO* strategy).

A similar pattern to the previous datasets can be seen in Table 7.6 with the computation times, though the difference between *RO* and the hypermutation strategies is much smaller in this experiment. This is likely due to the similar (median) number of clusters obtained for the methods, as shown in Figure 7.10. The *HV* trigger mechanism once again saw a reduction in the computation time, but noticeably less when compared to the time reductions seen with the *HK* and *UKC* datasets. This is due to the poorer performance requiring a higher average

Table 7.6: Normalized computation time mean and standard deviation for HAWKS datasets

Method	*Random*	*Interval*	*HV*
CO	0.377 ± 0.088	0.399 ± 0.065	0.367 ± 0.081
FM	0.440 ± 0.100	0.458 ± 0.075	0.408 ± 0.092
RO	0.433 ± 0.096	0.443 ± 0.073	0.407 ± 0.087
TH_{all}	0.454 ± 0.099	0.475 ± 0.079	0.423 ± 0.097
TH_{new}	0.452 ± 0.099	0.479 ± 0.075	0.430 ± 0.093
Δ-MOCK ($sr5$)		0.763 ± 0.097	
Δ-MOCK ($sr1$)		0.096 ± 0.038	

number of triggers (thus also validating that the reference gradient does correctly reflect a difference in the performance), which was 3.24 times across the HAWKS datasets[6]. The restriction not to trigger more than $T-1$ times in order to ensure a fair comparison does lessen the flexibility of the *HV* approach, which would not exist in normal use. The need to calculate a new reference gradient, blocking further triggers, does inherently limit the number of times this approach can change δ, however.

At least for the given hyperparameters, this experiment serves to confirm the potential difficulty of datasets that HAWKS can generate and the vast importance of δ to the performance of this algorithm. As these datasets are not particularly large, yet still required a lower δ value, there appears to be a higher proportion of relevant edges that are important in the MST. This may be due to a closer structure of the datasets, though from Section A.5.1 we know that the average silhouette width of the *HK* datasets is also quite low, yet a δ value corresponding to $sr1$ was far more effective on those datasets.

7.3.2 Results: Instance space

The boxplots provided in Figure 7.9 give an overall perspective of performance between the methods, but do not provide finer-grained information about the nature of the different algorithms, and whether different properties resulted in

[6]Compared to 1.72 and 2.13 with the *HK* and *UKC* datasets respectively.

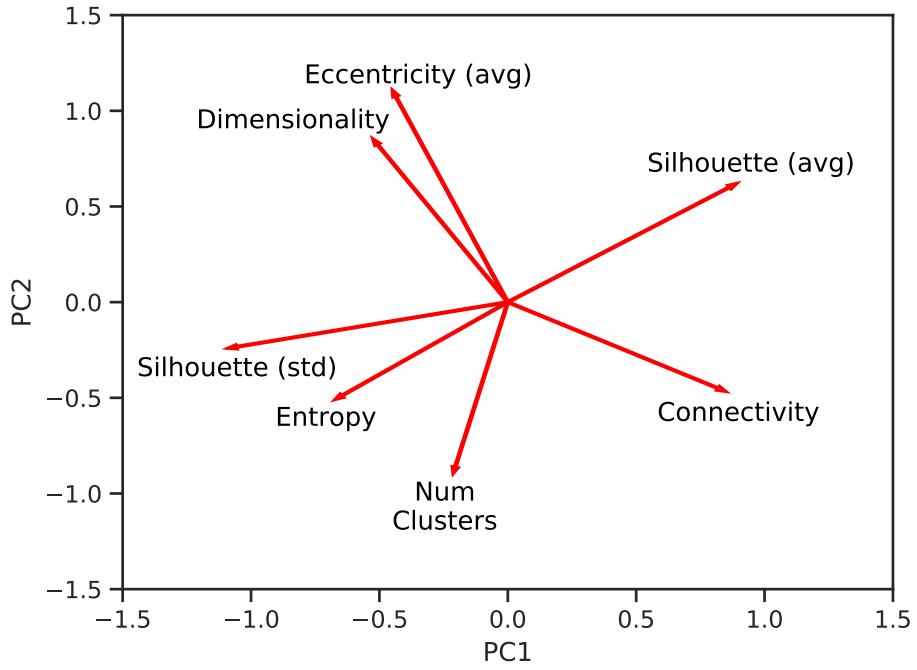


Figure 7.11: The direction and magnitude of the problem features as they contribute to the first two principal components.

different performance between the methods. To obtain this information, we use the problem feature set designed in Section 6.2.

Throughout this section, we select only results that use the *HV* trigger mechanism, so that we can further analyze our proposed method. The 7 problem features (defined in Section 6.2.1) were calculated for the 378 datasets (350 *HK*, 20 HAWKS, and 8 *UKC*) as before. The components for this projection can be seen in Figure 7.11, which shows a more directional spread of the features than previously obtained (in Figure 6.9). To see how each problem feature varies across the space, visualizations of the instance space can be found in Section B.1.3. The two principal components account for 65.88% of the variance, and so there is some information loss in this projection, but less compared to that previously seen (which was 57.56%).

The instance space itself, with the datasets themselves highlighted by the different sources that they originate from are shown in Figure 7.12. There is a clear separation between the three sets of datasets, indicating that they each have notably different properties compared to each other in this space. To further investigate the descriptive power (in terms of the performance of clustering algorithms), and to see how the methods compare in more fine-grained detail, we

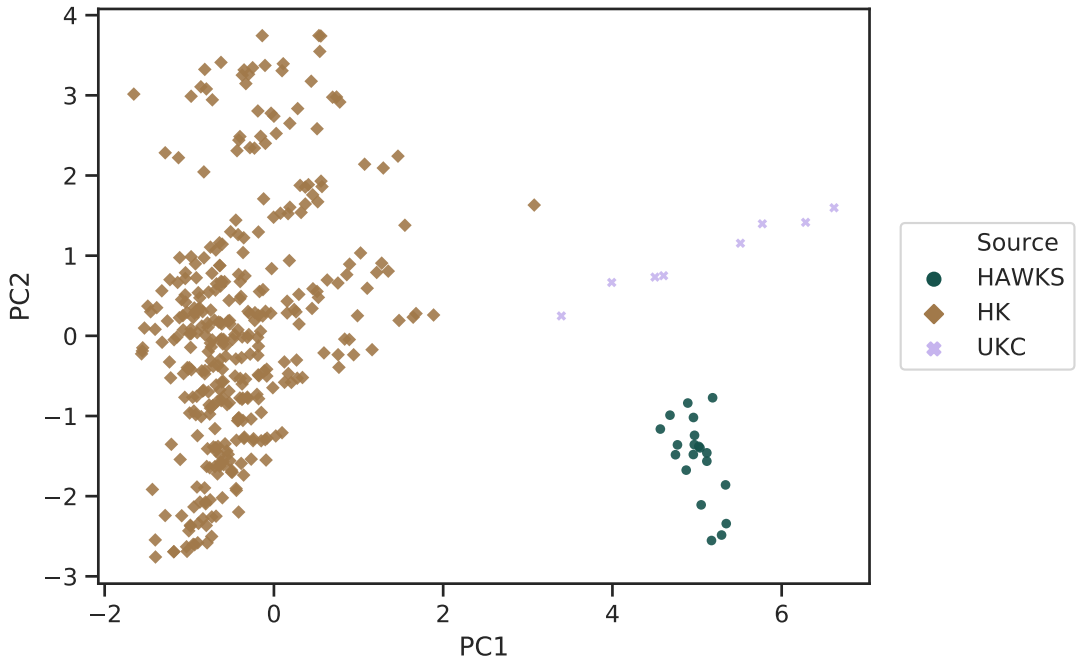


Figure 7.12: The instance space of the datasets with each of the three dataset sources (HAWKS, *HK*, and *UKC*) highlighted.

can modify the instance space to highlight which approaches performed best.

Figure 7.13 shows the instance space again, but with the 'winning' method highlighted. As each method was run 30 times on each dataset, we must first decide how to select a single value (ARI) to use to compare performance. For this, we select the median value to provide a view of the *average* performance of the methods (note that the instance space using the maximum ARI found for each method is shown in Section B.1.1). This approach is not too dissimilar with the previous stochastic clustering algorithms (GMM and K-Means++), which themselves have been run several times for each dataset, taking the 'best' run from each. The difference is that the 'best' in that scenario is the minimum lower bound of the log-likelihood (for GMM) or the minimum within-cluster sum of squares (for K-Means++), whereas for MOCK we are *directly* evaluating using the true labels to select a solution, primarily due to the open-problem nature of selecting a solution from the Pareto front. Thus, while this method prevents us from directly (and meaningfully) comparing results between these algorithms, within the variants of MOCK discussed in this chapter as long as our method of model selection remains constant we can compare results. For ease, the results

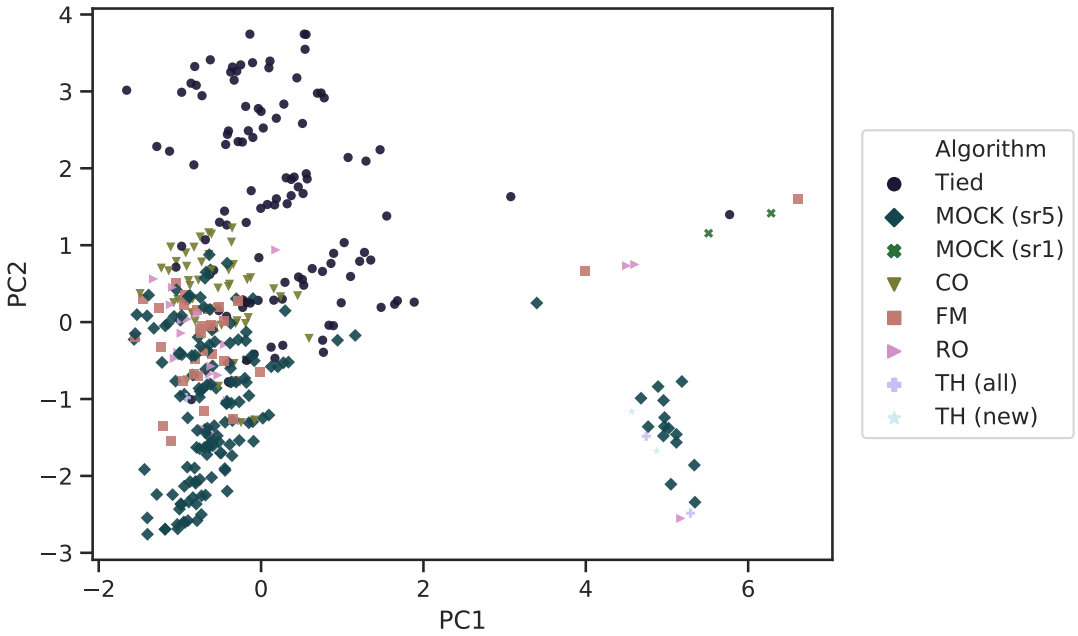


Figure 7.13: The instance space of the datasets highlighted according to the method that achieved the best performance (measured by the median ARI of the best individuals from the 30 runs).

Table 7.7: Number of 'wins' for each approach across the 378 datasets

Method	Number of 'wins'
Δ-MOCK ($sr5$)	155
Tied	109
CO	47
FM	33
RO	24
TH_{all}	4
TH_{new}	3
Δ-MOCK ($sr1$)	3

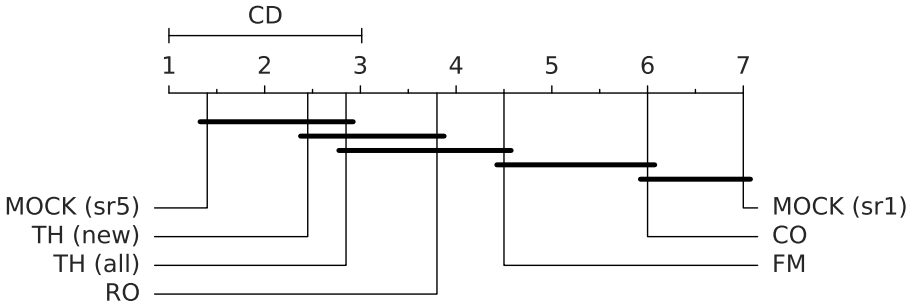
have been tabulated in Table 7.7[7].

For many of the datasets, there was a tie between two or methods and thus no definitive winner. These have been categorized as "Tied", and generally represent the easier datasets where multiple approaches could get a maximum ARI of 1. In general there is a mixture of performance, apart from a group of *HK* datasets (in the bottom left of the instance) where Δ-MOCK ($sr5$) uniquely performs the best. According to the problem features, this region has the datasets with the lowest silhouette width, indicating that for these less structured problems the search strategies may be more disruptive than helpful, and the larger search space is helpful (as the components are typically less homogeneous with regards to their cluster membership with such datasets).

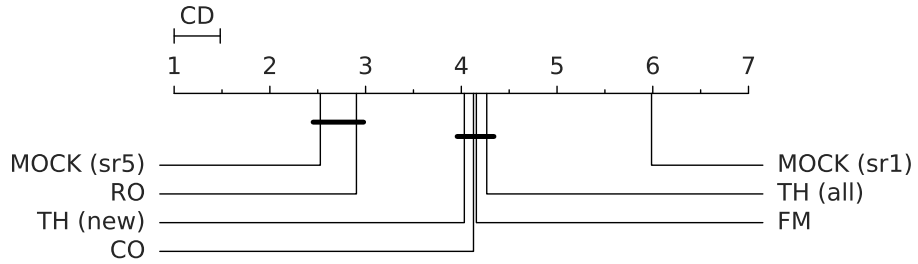
Predominantly, as expected, Δ-MOCK ($sr5$) is the best performing algorithm across the space. We see a much greater diversity of performance for the *UKC* datasets, though with the high performance of all approaches it is unlikely that this has a meaningful cause. Of interest is the centralized pocket of datasets for which *CO* is superior. Further investigating these datasets, they generally have a higher K, indicating that allowing the progressive enlargement of the search space was useful, but that the existing operators were sufficient to explore the new space without adding pressure to either objective (via the search strategies). For completeness, in Section B.1.2 we show the ARI across the space for each of the 7 methods (two baselines and five search strategies), which shows a clear gradient of performance across the space for each of the methods, which is particularly clear with the *HK* datasets. This is encouraging for tackling the ASP (i.e. predicting the performance according to the problem features), as no correlation between the projection and the performance would indicate that the problem features do not reflect difficulty.

We can also use the critical difference (CD) plots presented in Section 5.2.4 to further analyze the performance of these methods across the datasets. Figure 7.14 shows the CD plots for the datasets from HAWKS (a) and *HK* (b); note that *UKC* is not shown here as the Friedman test failed to show a significant difference between the approaches. This is likely due to the differences being too slight, as seen in Figure 7.5 and Table 7.3, and too few datasets being used to tease out further differences.

[7]Note that the lower number of wins for *RO*, when compared to methods such as *FM* which it has generally been shown to be superior to, is to the equivalent performance with Δ-MOCK $sr5$ in the "Tied" category.



(a) HAWKS



(b) *HK*

Figure 7.14: Critical difference (CD) diagrams for HAWKS and *HK*, showing the average rank for each method across the datasets. Solid lines connect methods which are not significantly different from each other according to a two-tailed Nemenyi test. *UKC* is not shown as there was no significant difference according to the Friedman test.

In Figure 7.14b we can see that the baseline, Δ-MOCK ($sr5$), was statistically equivalent to the *RO* search strategy. This provides a different perspective from Figure 7.4c, which found *RO* to be worse than the baseline, though this included data from *each* of the 30 runs across all of the datasets, whereas here we just select a single value across the 30 runs for each dataset. There is also a difference in the statistical tests being used, as the Friedman test considers the *ranks* of the different approaches, whereas the Wilcoxon signed-rank test considers the magnitude of difference between the performance. As a result, we cannot fully conclude the relative efficacy of *RO* against the baseline. As the boxplots include more information from the *actual* runs, however, as opposed to an aggregation, that is arguably a better reflection of the *general* performance of the methods. Nonetheless, the much closer proximity of *RO* to the baseline than to the other methods indicates its efficacy on these datasets. Aside from Δ-MOCK ($sr1$), which performed far worse than any other approach, the remaining methods are equivalent. This highlights the aforementioned weaknesses of increasing the mutation rate (at least in general on these datasets, as we know it can be useful when K is higher).

For the HAWKS datasets, Figure 7.14a shows a more complex perspective of performance. As each solid line connects methods that can be considered equivalent under the Nemenyi two-tailed test, we can see many subgroups of methods that performed equivalently to each other, but as a whole worse than those unconnected with a higher rank (i.e. towards the left), and better than those unconnected with a lower rank. As previously observed (Table 7.5), the hypermutation strategies were the best performing on the HAWKS datasets, owing to the high number of clusters ($K = 100$) which MOCK underestimated (Figure 7.10). In general, the CD plot for these results is consistent with previous results.

7.3.3 Experimental summary

It is not conclusive that our method was overall superior to the original non-adaptive Δ-MOCK, though the differences were not large in absolute terms and were complemented by a significant reduction in computation time. The need to adjust the δ_{Low} and δ_{High} values highlight a primary downside of this approach: it is semi-adaptive only. While it may not be possible to circumvent the need for an initial δ value, in a fully adaptive scenario (where δ can be both increased and decreased, at a magnitude based on performance rather than stepwise) this is less

impactful on performance and thus reduces the need for multiple runs to tune these hyperparameters. The incorporation of these improvements is non-trivial, however, and will require extensive further work (which we discuss further in Section 8.2.3). Nonetheless, large reductions in computation time were observed for mostly equivalent (in absolute terms) performance, in an experimental setup that was difficult for Adaptive-MOCK (as in general the baseline was given a good value of δ, and our adaptive version started with a poor δ).

In this section, we used datasets from HAWKS to provide further insights into the trigger mechanism and search strategies, but also to Δ-MOCK in general. The datasets we used were generated specifically to be eccentric, and are difficult both due to this and the close proximity of clusters (these datasets have a silhouette width of between 0.55 and 0.60). This presented a scenario where the hypermutation strategies were advantageous (in contrast to the *HK* datasets), as the final population in all approaches had a median K considerably *under* the true value. Further experiments with different hyperparameters of Δ- and Adaptive-MOCK would be required to see if better performance can be obtained on these datasets, or if they just present properties (either deliberate, or through a lack of separability) that are inherently difficult. Despite the multi-objective nature of MOCK, it is predicated on the idea that the MST is able to capture all relevant information useful for clustering.

A notable omission is the use of the 'versus mode' (Section 6.3) of HAWKS to directly pit Δ-MOCK and Adaptive-MOCK against each other. While this could help identify if they have weaknesses with respect to each other, therein lies an issue that we have not previously needed to address: selecting a solution from the final population. A major advantage of evolutionary clustering is the generation of a population of solutions, which as we discussed for MOCK consists of solutions of varying K. Without *a priori* knowledge of the true K, selection of a single individual from this population is an open question in and of itself (Handl and Knowles 2007). For our results, we have used the labels to select the best individual so that we can measure the potential of the method, but as the diversity is a component of performance (and in practice labels are not available), this may not be the best approach to use in the 'versus mode'.

7.4 Summary

In this chapter we modified the Δ-MOCK algorithm (creating a variant we name "Adaptive-MOCK") to adjust the search space based on the current performance, as measured by the hypervolume. We compared multiple different search strategies with the goal of rapidly searching the new decision variables in the genotype, aiming to avoid repeated evaluations on solutions we have previously seen. Our hypervolume-based approach was successful in limiting the number of adaptations required, seeing a greater reduction in computation time for equivalent performance when compared to baseline methods that randomly decided when to increase the search space. Similarly, our search strategies were all able to more rapidly explore the search space, as compared to a baseline strategy (CO) that did not change anything after an increase in the search space. The use of datasets from HAWKS provided a different perspective, showing that some of the search strategies did have uses for datasets with different properties (to those of HK). Additionally, the use of our problem feature set constructed an instance space that highlighted some potential differences between the search strategies.

Chapter 8

Conclusion

This thesis has investigated the utility of evolutionary algorithms applied to cluster analysis, specifically for generating datasets useful for evaluating clustering algorithms and as a multi-objective clustering algorithm itself. The main findings of this thesis with reference to the original contributions (stated in Section 1.3) have been outlined in Section 8.1. In Section 8.2 we discuss some of the limitations of our work, and the various potential future avenues of research that our work has presented.

8.1 Contributions

Development of a synthetic data generator that enables the parametrization of different cluster properties in order to modify the difficulty of the datasets

In Chapter 5 we introduced HAWKS, a synthetic data generator that used an evolutionary algorithm to optimize and explore different cluster properties that could be set by the user to create datasets of varied difficulty. This was achieved by setting a target value for the silhouette width, which is an internal cluster validity index (defined in Section 2.5.1) that takes into account both the intra-cluster compactness and inter-cluster separation. To ameliorate the difficulty and avoid issues that arise due to the silhouette width being an average over data points, two constraints have been added. The first measures the overlap as a percentage of data points whose nearest neighbour is in a different cluster, and the second measures the cluster eccentricity via the ratio of the largest to

smallest principal axes (i.e. the eigenvalues of the covariance matrix). To embed a preference between satisfying these constraints and the objective, which is useful to generate challenging datasets when these parameters conflict, the stochastic ranking was used.

Comparison of existing synthetic data generators for clustering in terms of the diversity of these datasets, measured by performance across a range of clustering algorithms

In Section 5.2.4, we generated a set of datasets using HAWKS by varying a few simple parameters such as the silhouette width target and number of clusters. The performance of four clustering algorithms (average-linkage, GMM, K-Means++, and single-linkage) on these datasets was compared to datasets produced by two popular generators: *HK* (Handl and Knowles 2005b) and *QJ* (Qiu and Joe 2006a). Greater diversity of performance (as measured by the average ranking of each algorithm) was found for the datasets produced by HAWKS. This was taken further in Section 6.2 where we compared more generators/sets of datasets with more diverse datasets from HAWKS (by varying the aforementioned constraints). Once again, a broader range of performance was seen across the clustering algorithms for the datasets produced by HAWKS.

Visualization of these datasets on an instance space using a set of informative features that describe different problem characteristics, in order to illustrate their diversity

In Section 5.2.4, and further developed in Section 6.2.4, we presented an instance space to visualize the datasets according to their problem features. In Section 6.2.1 we designed a more descriptive set of problem features (going beyond aspects such as the number of clusters) to take into account properties such as the average eccentricity and entropy of cluster sizes, enabling the construction of a more informative instance space. The utility of this problem feature set for describing the performance of clustering algorithms has been further validated in Figure B.2 by the correlation of performance (of Δ- and Adaptive-MOCK) across the instance space.

Identification of areas in this instance space without datasets, and the generation of datasets with properties to fill this area

In Section 6.2.5 we have specifically tried to generate datasets that were different (according to their problem features) from those previously produced, motivated by the clear separation of datasets produced by *HK* and HAWKS. Through a small modification of HAWKS' parameters we have been able to generate datasets in a different area of the instance space, with some datasets more similar to *HK* and others more distinct to any previously seen.

Extension of the generator to produce datasets that directly maximize the performance difference between two clustering algorithms

In Section 6.3 we modified the objective function of HAWKS to directly maximize the performance difference of two clustering algorithms. This removes the need for careful parameterization in order to create different challenges for different algorithms, thereby directly generating datasets that exploit an algorithm's relative weakness(es). We generated datasets for each of the aforementioned four clustering algorithms, with each algorithm both as the 'winner' and 'loser' (in terms of their performance being maximized and minimized, respectively). We found a surprisingly nuanced capability of HAWKS to generate datasets that directly found and exploited the weaknesses of each algorithm, which can be useful for further understanding and developing cluster algorithms.

Creation of a more flexible encoding for evolutionary clustering through the extension of an existing algorithm (Δ-MOCK) to reduce computation time and the importance of dataset-specific hyperparameterization

In Chapter 7 we used an existing evolutionary algorithm (Δ-MOCK) that generates partitions of data and took a step towards addressing its major limitation: scalability. The flexibility in clustering provided by this algorithm comes at the cost of a *vast* search space that, despite its specialized initialization scheme and other well-designed components, does not allow practical application of the algorithm to very large datasets (e.g. $N > 1{,}000{,}000$). In Section 7.2.2 we proposed Adaptive-MOCK, which begins with a small search space and iteratively enlarges it based on the rate of convergence (as measured by the hypervolume). We explored the use of different (search) strategies to rapidly explore the new search

space. In general, across the datasets studied, equivalent performance is obtained for a vastly lower computation time (between half and two-thirds that of the original Δ-MOCK). We then applied HAWKS to gain further insights into the differences between the search strategies and their applicability for datasets with different properties.

8.2 Future Work

This work opens up many possible avenues of future investigation. We first discuss more over-arching future work relating the algorithm selection problem (ASP) and clustering, followed by separate sections that discuss specific work on both HAWKS and Adaptive-MOCK.

8.2.1 Tackling the ASP for clustering

In Section 4.2, we discussed (at length) the nature of the ASP and how it relates to other fields of research (such as meta-learning). Our work in Chapters 5 and 6 focussed on developing a descriptive problem feature set and flexible generator in order to have the information necessary to predict the performance of different clustering algorithms on different datasets.

Naturally, the next step is to take our developments with HAWKS and actually tackle the ASP directly for clustering. While this poses many further questions of methodology for learning the mapping between the problem features and algorithmic performance, we have discussed previous work (in Section 4.2.1) that has attempted this, highlighting its feasibility. Recent work in areas such as exploratory landscape analysis, algorithm configuration, and meta-learning may be useful in this regard.

8.2.2 HAWKS

Various aspects of HAWKS can be modified and improved going forward. Its modular construction allows for these to be pursued independently of one another, yet the improvements can be shared.

Representation

An obvious issue with HAWKS that has been highlighted is its singular use of a Gaussian representation of clusters. While this provides useful properties (such as designing a geometrically interpretable mutation operator), it creates an inherent predisposition to particular cluster shapes (i.e. convex clusters) regardless of eccentricity.

Using a different distribution is unlikely to fully address this issue, and any methods of transformation/embedding to adjust the cluster shapes can add additional complexity (and potential redundancy) to the representation, increasing the difficulty of the optimization process. Nonetheless, a greater diversity of clusters can assist with the construction of a broader range of datasets.

This is related to a capability that the *QJ* (Qiu and Joe 2006a) generator exhibited that HAWKS did not — noisy features. While HAWKS is able to perturb the covariance matrix, members of each cluster are sampled from a Gaussian with this covariance. The introduction of uncorrelated or noisy features, either as additional elements of the genotype or through modification of the Gaussian itself, will add an additional aspect of difficulty to the generator. This is particularly important when considering clustering algorithms that have inherent feature selection (such as methods that integrate clustering with neural networks [Yang et al. 2017]).

Versus mode

Generating datasets that maximize the performance difference between algorithms (Section 6.3) is a useful way to understand the mechanistic differences between algorithms without having to first identify datasets that can tease out these properties (which, if unknown, is an impossible task). At present, this has been presented as a single-objective approach between two algorithms. It is easy to envisage the expansion of this, both in terms of the number of algorithms and potentially number of objectives. In the former case, having multiple 'losing' algorithms requires identification of properties that the 'winning' algorithm is more *distinctly* able to capture, further refining the ability of the generator to explore the unique strengths of a single algorithm. In the latter case, the formulation of multiple objectives (such as a two-objective problem where the performance difference of two algorithms is simultaneously maximized) permits exploration of the trade-off between the two algorithms over a more continuous scale.

Further development of this approach can then permit deeper analysis of the resulting datasets' problem features to directly quantify the differences between the algorithms. The integration of the instance space with this work can better help to establish algorithmic 'footprints' and thus the algorithm selection problem itself.

Eccentricity in higher dimensions

The generation of eccentric clusters in higher dimensions is a non-trivial task, as highlighted by the design in Handl and Knowles (2005b) of a separate generator specialized for this task. The *average eccentricity* problem feature we designed in Section 6.2.1 specifically to measure this highlighted that the *HK* generator was notably more successful in this regard. Although the initialization scheme of HAWKS can encourage more eccentric clusters, and the λ^{ratio} constraint can also encourage this, the covariance mutation operator may be unsuitable at higher dimensions to create high eccentricity in a reasonable number of generations. More consideration in the initialization stage of generating individuals that better adhere to the λ^{ratio} constraint may better generate a wider array of eccentricities among the individuals in higher dimensions.

Benchmark construction

The experiment in Section 6.2 took a further step towards the creation of a more comprehensive set of datasets that can be used to benchmark clustering algorithms. Robust comparison of the diversity for existing clustering datasets (similar to Macià and Bernadó-Mansilla [2014] but for clustering instead of classification) is lacking, partially due to the abundance of such datasets. The aforementioned improvements to HAWKS, on top of the study of diversity already performed in Section 6.2, should further facilitate and encourage the study of these datasets and subsequent construction of a more comprehensive benchmark suite.

The instance space and problem features are vital in the construction of a benchmark suite, as they allow for identification of where there may be datasets that lack particular properties or if there is a lack of diversity in an algorithm's footprint (for any competent algorithm, there should be datasets where it performs the best). Throughout this thesis we have used PCA to project our problem

features down to 2D for the instance space, but previously discussed work (Kandanaarachchi, Muñoz, and Smith-Miles 2019) indicates that other methods of projection may help to improve the utility of the visualization.

At a similar time to Shand et al. (2019), Iglesias et al. (2019) released a generator (named *MDCGen*) that aims to address the deficiencies of the generators discussed in Section 4.3.2. It shares some conceptual similarities and aims with HAWKS, though is quite different mechanistically. They outline a list of 16 requirements that a generator must satisfy (specifically for generating datasets for clustering), ranging from control over distributions, overlap, outliers etc. to a reproducible tool. Note that, while they do not state it is a requirement, we believe an additional requirement should be that the generator is open-source and easily-configurable.[1]

In contrast to HAWKS, *MDCGen* does not use optimization to position clusters such that they meet certain criteria. This is achieved primarily through the use of equidistant hyperplanes to create a grid upon which the clusters are placed (which are then scaled and transformed later according to the configuration). The overlap between clusters obtained is then evaluated using the silhouette width (Section 2.5.1), thereby giving an indication of whether the point should (or should not) be placed in that cluster (as opposed to Handl and Knowles [2005b] which compares the *true* labels between nearest neighbours). At present, there appear to be no experiments which give an indication into the range of performance that can be obtained on the datasets produced by *MDCGen* by varying its parameters, making it difficult to verify how effective their grid-based approach to placing clusters is. Considering the properties and potential power of this generator, a direct comparison (both qualitative and using the instance space) with HAWKS would be informative.

8.2.3 Adaptive-MOCK

In Section 7.4 we discussed some of the issues with our approach and therefore potential areas for further work. In this section we provide further details about such work.

[1] Both HAWKS and *MDCGen* (available at `https://github.com/CN-TU/mdcgen-matlab`) satisfy this.

Adaptation

The vast reduction in computation time is a clear advantage of using an adaptive encoding, though the real utility of this reduction is seen when there is no *a priori* knowledge of a suitable δ. In such scenarios, our method of gradually decreasing δ from a *very* high value may also fail if the other hyperparameters (such as the pre-defined number of resolution levels, T) are incorrect, or the initial δ_{High} is too restrictive to be useful. Combined with the limited ability of δ to only be decreased, and not increased, there is an obvious need for improvement.

The use of the hypervolume as a method to track performance is valid, but as it is measuring whether the search has (mostly) *converged*, this does not facilitate identifying when δ should be *increased*. Thus, in the absence of an easy way to track performance it follows that we should use *self-adaptive* rather than *adaptive* approaches (Section 3.4.1). One such approach would be to define δ on an individual level, thus creating a population of mixed resolutions. A primary barrier to this approach is the inherent conflict between this and the "delta-evaluation" (Section 7.2.2), potentially reducing the vast computational savings of Δ-MOCK. Preliminary work (not shown in this thesis) supports the potential for a more flexible approach, however.

Components & mutation bias

The mutation bias illustrated in Section 7.2.6 presents an issue for both Adaptive-MOCK and Δ-MOCK itself. It highlights a more fundamental issue with the operator — the use of the degree of interestingness (DI) in Δ-MOCK was aimed at identifying which edges in the minimum spanning tree (MST) were not important to the search, yet the mutation does not make an equivalent adjustment. The δ hyperparameter transforms the search into finding the optimal combination of *components* (rather than individual points) which are ideally homogeneous groups of data points. Thus, this shift of searching for the optimal configuration of components should be mirrored in the mutation operator, as the present version of searching through each point's L nearest neighbours may not affect the fitness.

Neighbourhood hyperparameter

While δ has the biggest impact on performance, the L neighbourhood hyperparameter will also have a significant impact, yet this is not considered for adaptation. Primarily, this was due to the complexity and interference with "delta-evaluation", which has the potential to significantly increase the computation required. Throughout this work, we use a fixed $L = 10$ as established by Handl and Knowles (2007), but recognize that this may not be ideal and that future work should consider the adaptation of *multiple* hyperparameters, not just δ.

Bibliography

Ackerman, Margareta, Shai Ben-David, and David Loker (2010). "Towards Property-Based Classification of Clustering Paradigms". In: *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*. Ed. by John D. Lafferty et al. Curran Associates, Inc., pp. 10–18. URL: http://papers.nips.cc/paper/4101-towards-property-based-classification-of-clustering-paradigms.

Adolfsson, Andreas, Margareta Ackerman, and Naomi C. Brownstein (2019). "To cluster, or not to cluster: An analysis of clusterability methods". In: *Pattern Recognit.* 88, pp. 13–26. DOI: 10.1016/j.patcog.2018.10.026.

Allmendinger, Richard and Joshua D. Knowles (2010). "Evolutionary Optimization on Problems Subject to Changes of Variables". In: Lecture Notes in Computer Science 6239. Ed. by Robert Schaefer et al., pp. 151–160. DOI: 10.1007/978-3-642-15871-1_16.

Allmendinger, Richard and Joshua D. Knowles (2013). "On Handling Ephemeral Resource Constraints in Evolutionary Search". In: *Evol. Comput.* 21.3, pp. 497–531. DOI: 10.1162/EVCO_a_00097.

Allmendinger, Richard et al. (2017). "Surrogate-assisted multicriteria optimization: Complexities, prospective solutions, and business case". In: *Journal of Multi-Criteria Decision Analysis* 24.1-2, pp. 5–24.

Anderberg, Michael R. (1973). *Cluster Analysis for Applications (Probability and mathematical statistics)*. Vol. 19. Academic Press. DOI: 10.1016/C2013-0-06161-0.

Ankerst, Mihael et al. (1999). "OPTICS: Ordering Points To Identify the Clustering Structure". In: *SIGMOD 1999, Proceedings ACM SIGMOD International*

Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA. Ed. by Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh. ACM Press, pp. 49–60. DOI: 10.1145/304182.304187.

Arbelaitz, Olatz et al. (2013). "An extensive comparative study of cluster validity indices". In: *Pattern Recognit.* 46.1, pp. 243–256. DOI: 10.1016/j.patcog.2012.07.021.

Arthur, David and Sergei Vassilvitskii (2007). "k-means++: The advantages of careful seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*. SIAM, pp. 1027–1035. URL: http://dl.acm.org/citation.cfm?id=1283383.1283494.

Bäck, Thomas (1992). "Self-adaptation in genetic algorithms". In: *Proceedings of the First European Conference on Artificial Life*. MIT Press, pp. 263–271.

Bäck, Thomas (1995). "Generalized Convergence Models for Tournament- and (mu, lambda)-Selection". In: *Proceedings of the 6th International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 15-19, 1995*. Ed. by Larry J. Eshelman. Morgan Kaufmann, pp. 2–8. URL: http://dl.acm.org/citation.cfm?id=645514.657771.

Bäck, Thomas, David B. Fogel, and Zbigniew Michalewicz (2000). *Evolutionary computation 1: Basic algorithms and operators*. CRC Press.

Bardenet, Rémi et al. (2013). "Collaborative hyperparameter tuning". In: *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. Vol. 28. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 199–207. URL: http://proceedings.mlr.press/v28/bardenet13.html.

Bayá, Ariel E. and Pablo M. Granitto (2013). "How Many Clusters: A Validation Index for Arbitrary-Shaped Clusters". In: *IEEE/ACM Trans. Comput. Biology Bioinform.* 10.2, pp. 401–414. DOI: 10.1109/TCBB.2013.32.

Bellman, Richard (1966). "Dynamic programming". In: *Science* 153.3731, pp. 34–37. DOI: 10.1126/science.153.3731.34.

Benavoli, Alessio, Giorgio Corani, and Francesca Mangili (2016). "Should We Really Use Post-Hoc Tests Based on Mean-Ranks?" In: *J. Mach. Learn. Res.* 17, 5:1–5:10. URL: http://jmlr.org/papers/v17/benavoli16a.html.

Ben-David, Shai (2018). "Clustering - What Both Theoreticians and Practitioners Are Doing Wrong". In: *Proceedings of the Thirty-Second AAAI Conference*

on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 7962–7964. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17420.

Bennett, David A., Ningchuan Xiao, and Marc P. Armstrong (2004). "Exploring the geographic consequences of public policies using evolutionary algorithms". In: *Annals of the Association of American Geographers* 94.4, pp. 827–847.

Beyer, Kevin et al. (1999). "When is "nearest neighbor" meaningful?" In: *International conference on database theory*. Springer, pp. 217–235.

Bezdek, James C. and Nikhil R. Pal (1995). "Cluster Validation with Generalized Dunn's Indices". In: *2nd New Zealand Two-Stream International Conference on Artificial Neural Networks and Expert Systems (ANNES '95), November 20-23, 1995, Dunedin, New Zealand*. IEEE Computer Society, pp. 190–193. DOI: 10.1109/ANNES.1995.499469.

Bezdek, James C. and Nikhil R. Pal (1998). "Some new indexes of cluster validity". In: *IEEE Trans. Systems, Man, and Cybernetics, Part B* 28.3, pp. 301–315. DOI: 10.1109/3477.678624.

Bilalli, Besim, Alberto Abelló, and Tomàs Aluja-Banet (2017). "On the predictive power of meta-features in OpenML". In: *Int. J. Appl. Math. Comput. Sci.* 27.4, pp. 697–712. DOI: 10.1515/amcs-2017-0048.

Birattari, Mauro (2009). *Tuning Metaheuristics - A Machine Learning Perspective*. Vol. 197. Studies in Computational Intelligence. Springer. ISBN: 978-3-642-00482-7. DOI: 10.1007/978-3-642-00483-4.

Bishop, Christopher M. (2007). *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer. ISBN: 9780387310732. URL: http://www.worldcat.org/oclc/71008143.

Blum, Christian and Andrea Roli (2003). "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM Comput. Surv.* 35.3, pp. 268–308. DOI: 10.1145/937503.937505.

Bolshakova, Nadia and Francisco Azuaje (2003). "Cluster validation techniques for genome expression data". In: *Signal processing* 83.4, pp. 825–833. DOI: 10.1016/S0165-1684(02)00475-9.

Bowker, Geoffrey C. and Susan Leigh Star (2000). *Sorting Things Out: Classification and its Consequences*. MIT Press.

Branke, Jürgen (2012). *Evolutionary Optimization in Dynamic Environments*. Vol. 3. Springer Science & Business Media. DOI: 10.1007/978-1-4615-0911-0.

Castiello, Ciro, Giovanna Castellano, and Anna Maria Fanelli (2005). "Meta-data: Characterization of Input Features for Meta-learning". In: *Modeling Decisions for Artificial Intelligence, Second International Conference, MDAI 2005, Tsukuba, Japan, July 25-27, 2005, Proceedings*. Vol. 3558. Lecture Notes in Computer Science. Springer, pp. 457–468. DOI: 10.1007/11526018_45.

Chacón, José E. and Ana I. Rastrojo (2020). *Minimum adjusted Rand index for two clusterings of a given size*. arXiv: 2002.03677 [stat.ML].

Chiang, Mark Ming-Tso and Boris G. Mirkin (2010). "Intelligent Choice of the Number of Clusters in K-Means Clustering: An Experimental Study with Different Cluster Spreads". In: *J. Classification* 27.1, pp. 3–40. DOI: 10.1007/s00357-010-9049-5.

Choi, Sung-Soon and Byung Ro Moon (2008). "Normalization for Genetic Algorithms With Nonsynonymously Redundant Encodings". In: *IEEE Trans. Evolutionary Computation* 12.5, pp. 604–616. DOI: 10.1109/TEVC.2007.913699.

Chugh, Tinkle et al. (2019a). "A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms". In: *Soft Comput.* 23.9, pp. 3137–3166. DOI: 10.1007/s00500-017-2965-0.

Chugh, Tinkle et al. (2019b). "Multiobjective shape design in a ventilation system with a preference-driven surrogate-assisted evolutionary algorithm". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. Ed. by Anne Auger and Thomas Stützle. ACM, pp. 1147–1155. DOI: 10.1145/3321707.3321745.

Cobb, Helen G. (1990). "An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments". In: *Technical Report*.

Coello Coello, Carlos A. (2002). "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art". In: *Computer methods in applied mechanics and engineering* 191.11-12, pp. 1245–1287.

Corne, David W. and Alan P. Reynolds (2010). "Optimisation and Generalisation: Footprints in Instance Space". In: *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*. Ed. by Robert Schaefer et al. Vol. 6238. Lecture Notes in Computer Science. Springer, pp. 22–31. DOI: 10.1007/978-3-642-15844-5_3.

Corne, David W. et al. (2001). "PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization". In: pp. 283–290. URL: https://dl.acm.org/doi/10.5555/2955239.2955289.

Davies, David L. and Donald W. Bouldin (1979). "A Cluster Separation Measure". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 1.2, pp. 224–227. DOI: 10.1109/TPAMI.1979.4766909.

De Jong, Kenneth Alan (1975). "An Analysis of the Behavior of a Class of Genetic Adaptive Systems". PhD thesis. Ann Arbor, MI, USA.

Deb, Kalyanmoy (2000). "An efficient constraint handling method for genetic algorithms". In: *Computer methods in applied mechanics and engineering* 186.2-4, pp. 311–338. DOI: 10.1016/S0045-7825(99)00389-8.

Deb, Kalyanmoy (2012). *Optimization for Engineering Design - Algorithms and Examples, Second Edition*. PHI Learning Private Limited. ISBN: 978-81-203-4678-9.

Deb, Kalyanmoy, Manikanth Mohan, and Shikhar Mishra (2005). "Evaluating the epsilon-Domination Based Multi-Objective Evolutionary Algorithm for a Quick Computation of Pareto-Optimal Solutions". In: *Evol. Comput.* 13.4, pp. 501–525. DOI: 10.1162/106365605774666895.

Deb, Kalyanmoy et al. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197. DOI: 10.1109/4235.996017.

Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1, pp. 1–22.

Demšar, Janez (2006). "Statistical comparisons of classifiers over multiple data sets". In: *Journal of Machine learning research* 7.Jan, pp. 1–30. URL: http://jmlr.org/papers/v7/demsar06a.html.

Dheeru, Dua and Efi Karra Taniskidou (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Ding, Chris H. Q. and Xiaofeng He (2004). "K-nearest-neighbor consistency in data clustering: incorporating local information into global optimization". In: *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, March 14-17, 2004*. Ed. by Hisham Haddad et al. ACM, pp. 584–589. DOI: 10.1145/967900.968021.

Duda, Richard O., Peter E. Hart, and David G. Stork (2001). *Pattern classification, 2nd Edition*. Wiley. ISBN: 9780471056690. URL: http://www.worldcat.org/oclc/41347061.

Dunn, J. C. (1973). "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". In: *Journal of Cybernetics* 3.3, pp. 32–57. DOI: 10.1080/01969727308546046. URL: https://doi.org/10.1080/01969727308546046.

Dunteman, George H. (1989). *Principal components analysis*. 69. Sage.

Edwards, A. W. F. and L. L. Cavalli-Sforza (1965). "A Method for Cluster Analysis". In: *Biometrics* 21.2, pp. 362–375. ISSN: 0006341X, 15410420. URL: http://www.jstor.org/stable/2528096.

Eiben, Ágoston E, Robert Hinterding, and Zbigniew Michalewicz (1999). "Parameter control in evolutionary algorithms". In: *IEEE Transactions on Evolutionary Computation* 3.2, pp. 124–141. ISSN: 1089-778X. DOI: 10.1109/4235.771166.

Eiben, Ágoston E. and Selmar K. Smit (2011). "Parameter tuning for configuring and analyzing evolutionary algorithms". In: *Swarm and Evolutionary Computation* 1.1, pp. 19–31. DOI: 10.1016/j.swevo.2011.02.001.

Eiben, Ágoston E. and James E. Smith (2015). *Introduction to Evolutionary Computing*. 2nd. Springer Publishing Company, Incorporated. ISBN: 978-3-662-44873-1. DOI: 10.1007/978-3-662-44874-8.

Emmerich, Michael T. M. and André H. Deutz (2018). "A tutorial on multiobjective optimization: fundamentals and evolutionary methods". In: *Nat. Comput.* 17.3, pp. 585–609. DOI: 10.1007/s11047-018-9685-y.

Eshelman, Larry J. (1990). "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination". In: *Proceedings of the First Workshop on Foundations of Genetic Algorithms. Bloomington Campus, Indiana, USA, July 15-18 1990*. Ed. by Gregory J. E. Rawlins. Morgan Kaufmann, pp. 265–283. DOI: 10.1016/b978-0-08-050684-5.50020-3.

Ester, Martin et al. (1996). "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*. Ed. by Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad. AAAI Press, pp. 226–231. URL: http://www.aaai.org/Library/KDD/1996/kdd96-037.php.

Everitt, Brian S. et al. (2011). *Cluster Analysis*. 5th. Wiley Publishing. ISBN: "978-0-470-74991-3".

Ferrari, Daniel Gomes and Leandro Nunes de Castro (2015). "Clustering algorithm selection by meta-learning systems: A new distance-based roblem characterization and ranking combination methods". In: *Inf. Sci.* 301, pp. 181–194. DOI: 10.1016/j.ins.2014.12.044.

Feurer, Matthias and Frank Hutter (2019). "Hyperparameter Optimization". In: *Automated Machine Learning - Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. The Springer Series on Challenges in Machine Learning. Springer, pp. 3–33. DOI: 10.1007/978-3-030-05318-5_1.

Fleetwood, Kelly (2004). "An introduction to differential evolution". In: *Proceedings of Mathematics and Statistics of Complex Systems (MASCOS) One Day Symposium, 26th November, Brisbane, Australia*, pp. 785–791.

Fogarty, Terence C. (1989). "Varying the Probability of Mutation in the Genetic Algorithm". In: *Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989*. Ed. by J. David Schaffer. Morgan Kaufmann, pp. 104–109.

Fogel, Lawrence J., Alvin J. Owens, and Michael J. Walsh (1966). "Artificial intelligence through simulated evolution". In:

Fonseca, Carlos M. and Peter J. Fleming (1996). "On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers". In: *Parallel Problem Solving from Nature - PPSN IV, International Conference on Evolutionary Computation. The 4th International Conference on Parallel Problem Solving from Nature, Berlin, Germany, September 22-26, 1996, Proceedings*. Ed. by Hans-Michael Voigt et al. Vol. 1141. Lecture Notes in Computer Science. Springer, pp. 584–593. DOI: 10.1007/3-540-61723-X_1022.

Fonseca, Carlos M. et al. (2011). "On the Computation of the Empirical Attainment Function". In: *Evolutionary Multi-Criterion Optimization - 6th International Conference, EMO 2011, Ouro Preto, Brazil, April 5-8, 2011. Proceedings*. Ed. by Ricardo H. C. Takahashi et al. Vol. 6576. Lecture Notes in Computer Science. Springer, pp. 106–120. DOI: 10.1007/978-3-642-19893-9_8. URL: https://doi.org/10.1007/978-3-642-19893-9%5C_8.

Fränti, Pasi and Sami Sieranoja (2018). "K-means properties on six clustering benchmark datasets". In: *Appl. Intell.* 48.12, pp. 4743–4759. DOI: 10.1007/s10489-018-1238-7.

Gallagher, Marcus (2019). "Fitness Landscape Analysis in Data-Driven Optimization: An Investigation of Clustering Problems". In: *IEEE Congress on Evolutionary Computation, CEC 2019, Wellington, New Zealand, June 10-13, 2019*. IEEE, pp. 2308–2314. DOI: 10.1109/CEC.2019.8790323.

García, Adán José and Wilfrido Gómez-Flores (2016). "Automatic clustering using nature-inspired metaheuristics: A survey". In: *Appl. Soft Comput.* 41, pp. 192–213. DOI: 10.1016/j.asoc.2015.12.001.

Garza-Fabre, Mario, Julia Handl, and Joshua D. Knowles (2017). "An Improved and More Scalable Evolutionary Approach to Multiobjective Clustering". In: *IEEE Transactions on Evolutionary Computation* 22.4, pp. 515–535. ISSN: 1089-778X. DOI: 10.1109/TEVC.2017.2726341.

Ghoreishi, Seyyedeh Newsha, Anders Clausen, and Bo Nørregaard Jørgensen (2017). "Termination Criteria in Evolutionary Algorithms: A Survey". In: *Proceedings of the 9th International Joint Conference on Computational Intelligence, IJCCI 2017, Funchal, Madeira, Portugal, November 1-3, 2017*. Ed. by Christophe Sabourin et al. SciTePress, pp. 373–384. DOI: 10.5220/0006577903730384.

Goldberg, David E. (1989). *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley. ISBN: 0-201-15767-5.

Goldberg, David E. (2002). "Genetic Algorithms and Innovation". In: *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Boston, MA: Springer US, pp. 1–9. ISBN: 978-1-4757-3643-4. DOI: 10.1007/978-1-4757-3643-4_1.

Goldberg, David E. and Kalyanmoy Deb (1990). "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms". In: *Proceedings of the First*

Workshop on Foundations of Genetic Algorithms. Bloomington Campus, Indiana, USA, July 15-18 1990. Ed. by Gregory J. E. Rawlins. Morgan Kaufmann, pp. 69–93. DOI: 10.1016/b978-0-08-050684-5.50008-2.

Grefenstette, John J. (1986). "Optimization of Control Parameters for Genetic Algorithms". In: *IEEE Trans. Systems, Man, and Cybernetics* 16.1, pp. 122–128. DOI: 10.1109/TSMC.1986.289288.

Guénoche, Alain, Pierre Hansen, and Brigitte Jaumard (1991). "Efficient algorithms for divisive hierarchical clustering with the diameter criterion". In: *Journal of classification* 8.1, pp. 5–30.

Guerrero, José Luis et al. (2009). "A stopping criterion based on Kalman estimation techniques with several progress indicators". In: *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*. Ed. by Franz Rothlauf. ACM, pp. 587–594. DOI: 10.1145/1569901.1569983.

Guyon, Isabelle, Ulrike Von Luxburg, and Robert C Williamson (2009). "Clustering: Science or art". In: *NIPS 2009 workshop on clustering theory*, pp. 1–11.

Hadka, David and Patrick M. Reed (2013). "Borg: An Auto-Adaptive Many-Objective Evolutionary Computing Framework". In: *Evol. Comput.* 21.2, pp. 231–259. DOI: 10.1162/EVCO_a_00075.

Hamida, S. Ben and Marc Schoenauer (2002). "ASCHEA: new results using adaptive segregational constraint handling". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*. Vol. 1. IEEE, pp. 884–889. DOI: 10.1109/CEC.2002.1007042.

Han, Jiawei, Micheline Kamber, and Jian Pei (2011). *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann. ISBN: 978-0123814791. URL: http://hanj.cs.illinois.edu/bk3/.

Handl, Julia and Joshua D. Knowles (2004). "Evolutionary multiobjective clustering". In: *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings*. Vol. 3242. Lecture Notes in Computer Science. Springer, pp. 1081–1091. DOI: 10.1007/978-3-540-30217-9_109.

Handl, Julia and Joshua D. Knowles (2005a). "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*. Ed. by Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler. Vol. 3410. Lecture Notes in Computer Science. Springer, pp. 547–560. DOI: 10.1007/978-3-540-31880-4_38.

Handl, Julia and Joshua D. Knowles (2005b). "Improvements to the scalability of multiobjective clustering". In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 3, pp. 2372–2379. DOI: 10.1109/CEC.2005.1554990.

Handl, Julia and Joshua D. Knowles (2006). "Feature subset selection in unsupervised learning via multiobjective optimization". In: *International Journal of Computational Intelligence Research* 2.3, pp. 217–238.

Handl, Julia and Joshua D. Knowles (2007). "An Evolutionary Approach to Multiobjective Clustering". In: *IEEE Transactions on Evolutionary Computation* 11.1, pp. 56–76. ISSN: 1089778X. DOI: 10.1109/TEVC.2006.877146.

Handl, Julia, Joshua D. Knowles, and Douglas B. Kell (2005). "Computational cluster validation in post-genomic data analysis". In: *Bioinform.* 21.15, pp. 3201–3212. DOI: 10.1093/bioinformatics/bti517.

Hansen, Nikolaus (2009). "Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed". In: *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009, Companion Material*. Ed. by Franz Rothlauf. ACM, pp. 2389–2396. DOI: 10.1145/1570256.1570333.

Hartigan, John A and Manchek A Wong (1979). "Algorithm AS 136: A k-means clustering algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1, pp. 100–108. DOI: 10.2307/2346830.

Hastie, Trevor, Robert Tibshirani, and Jerome H. Friedman (2009). "The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition". In: Springer Series in Statistics. DOI: 10.1007/978-0-387-84858-7.

Hemert, Jano I. van (2006). "Evolving Combinatorial Problem Instances That Are Difficult to Solve". In: *Evolutionary Computation* 14.4, pp. 433–462. DOI: 10.1162/evco.2006.14.4.433.

Hennig, Christian (2015). "What are the true clusters?" In: *Pattern Recognit. Lett.* 64, pp. 53–62. DOI: 10.1016/j.patrec.2015.04.009.

Hinterding, Robert, Zbigniew Michalewicz, and Ágoston E. Eiben (1997). "Adaptation in evolutionary computation: A survey". In: *Proceedings of 1997 Ieee International Conference on Evolutionary Computation (ICEC'97)*. IEEE, pp. 65–69. DOI: 10.1109/ICEC.1997.592270.

Ho, Pei Yee and Kazuyuki Shimizu (2007). "Evolutionary constrained optimization using an addition of ranking method and a percentage-based tolerance value adjustment scheme". In: *Inf. Sci.* 177.14, pp. 2985–3004. DOI: 10.1016/j.ins.2007.01.011.

Holland, John H. (1962). "Outline for a Logical Theory of Adaptive Systems". In: *J. ACM* 9.3, pp. 297–314. DOI: 10.1145/321127.321128.

Holland, John H. (1975). "Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control and artificial intelligence". In: *Ann Arbor: University of Michigan Press, 1975*.

Holland, John H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press. DOI: 10.7551/mitpress/1090.001.0001.

Holm, Sture (1979). "A simple sequentially rejective multiple test procedure". In: *Scandinavian journal of statistics*, pp. 65–70.

Hooker, John N. (1995). "Testing heuristics: We have it all wrong". In: *Journal of Heuristics* 1.1, pp. 33–42. DOI: 10.1007/BF02430364.

Houle, Michael E. et al. (2010). "Can Shared-Neighbor Distances Defeat the Curse of Dimensionality?" In: *Scientific and Statistical Database Management, 22nd International Conference, SSDBM 2010, Heidelberg, Germany, June 30 - July 2, 2010. Proceedings*. Ed. by Michael Gertz and Bertram Ludäscher. Vol. 6187. Lecture Notes in Computer Science. Springer, pp. 482–500. DOI: 10.1007/978-3-642-13818-8_34.

Huber, Peter J. (1985). "Projection Pursuit". In: *The Annals of Statistics* 13.2, pp. 435–475. DOI: 10.1214/aos/1176349519.

Hubert, Lawrence (1974). "Approximate evaluation techniques for the single-link and complete-link hierarchical clustering procedures". In: *Journal of the American Statistical Association* 69.347, pp. 698–704.

Hubert, Lawrence and Phipps Arabie (1985). "Comparing partitions". In: *Journal of classification* 2.1, pp. 193–218.

Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown (2011). "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*. Ed. by Carlos A. Coello Coello. Vol. 6683. Lecture Notes in Computer Science. Springer, pp. 507–523. DOI: 10.1007/978-3-642-25566-3_40.

Iglesias, Félix et al. (2019). "MDCGen: Multidimensional Dataset Generator for Clustering". In: *J. Classification* 36.3, pp. 599–618. DOI: 10.1007/s00357-019-9312-3.

Ishibuchi, Hisao, Noritaka Tsukamoto, and Yusuke Nojima (2008). "Evolutionary many-objective optimization: A short review". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1-6, 2008, Hong Kong, China*. IEEE, pp. 2419–2426. DOI: 10.1109/CEC.2008.4631121.

Jain, A. K. (2010). "Data clustering: 50 years beyond K-means". In: *Pattern recognition letters* 31.8, pp. 651–666. DOI: 10.1016/j.patrec.2009.09.011.

Jain, Anil K., M. Narasimha Murty, and Patrick J. Flynn (1999). "Data Clustering: A Review". In: *ACM Comput. Surv.* 31.3, pp. 264–323. DOI: 10.1145/331499.331504.

Jain, Brijnesh J., Hartmut Pohlheim, and Joachim Wegener (2001). "On termination criteria of evolutionary algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers San Francisco, California, USA, p. 768.

Jin, Yaochu (2011). "Surrogate-assisted evolutionary computation: Recent advances and future challenges". In: *Swarm Evol. Comput.* 1.2, pp. 61–70. DOI: 10.1016/j.swevo.2011.05.001.

Jing, Liping, Michael K. Ng, and Joshua Zhexue Huang (2007). "An Entropy Weighting k-Means Algorithm for Subspace Clustering of High-Dimensional Sparse Data". In: *IEEE Trans. Knowl. Data Eng.* 19.8, pp. 1026–1041. DOI: 10.1109/TKDE.2007.1048.

Joines, Jeffrey A. and Christopher R. Houck (1994). "On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GA's". In: *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Orlando, Florida, USA, June 27-29, 1994*. IEEE, pp. 579–584. DOI: 10.1109/ICEC.1994.349995.

Jong, Kenneth A. De and Jayshree Sarma (1992). "Generation Gaps Revisited". In: *Proceedings of the Second Workshop on Foundations of Genetic Algorithms. Vail, Colorado, USA, July 26-29 1992*. Ed. by L. Darrell Whitley. Morgan Kaufmann, pp. 19–28. DOI: 10.1016/b978-0-08-094832-4.50007-6.

Kandanaarachchi, Sevvandi, Mario A. Muñoz, and Kate Smith-Miles (2019). "Instance Space Analysis for Unsupervised Outlier Detection". In: *Evaluation and Experimental Design in Data Mining and Machine Learning 2019*. Vol. 2436. CEUR Workshop Proceedings, pp. 32–41. URL: http://ceur-ws.org/Vol-2436/article_4.pdf.

Karafotias, Giorgos, Mark Hoogendoorn, and Ágoston E. Eiben (2015). "Parameter Control in Evolutionary Algorithms: Trends and Challenges". In: *IEEE Trans. Evolutionary Computation* 19.2, pp. 167–187. DOI: 10.1109/TEVC.2014.2308294.

Kaufman, Leonard and Peter J. Rousseeuw (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley. ISBN: 978-0-47187876-6. DOI: 10.1002/9780470316801.

Kazarlis, Spiridon A. and Vassilios Petridis (1998). "Varying Fitness Functions in Genetic Algorithms: Studying the Rate of Increase of the Dynamic Penalty Terms". In: *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*. Ed. by A. E. Eiben et al. Vol. 1498. Lecture Notes in Computer Science. Springer, pp. 211–220. DOI: 10.1007/BFb0056864.

Kennedy, James (2010). "Particle Swarm Optimization". In: ed. by Claude Sammut and Geoffrey I. Webb, pp. 760–766. DOI: 10.1007/978-0-387-30164-8_630.

Kennedy, James and Russell Eberhart (1995). "Particle swarm optimization". In: *Proceedings of International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, November 27 - December 1, 1995*. IEEE, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

Kerschke, Pascal and Mike Preuss (2017). "Exploratory landscape analysis: advanced tutorial at GECCO 2017". In: *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings*. Ed. by Peter A. N. Bosman. ACM, pp. 762–781. DOI: 10.1145/3067695.3067696.

Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi (1983). "Optimization by simulated annealing". In: *Science* 220.4598, pp. 671–680. DOI: 10.1126/science.220.4598.671.

Kleinberg, Jon M. (2002). "An Impossibility Theorem for Clustering". In: *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*. Ed. by Suzanna Becker, Sebastian Thrun, and Klaus Obermayer. MIT Press, pp. 446–453. URL: http://papers.nips.cc/paper/2340-an-impossibility-theorem-for-clustering.

Korenius, Tuomo, Jorma Laurikkala, and Martti Juhola (2007). "On principal component analysis, cosine and Euclidean measures in information retrieval". In: *Information Sciences* 177.22, pp. 4893–4905. DOI: 10.1016/j.ins.2007.05.027.

Kotthoff, Lars et al. (2015). "Improving the State of the Art in Inexact TSP Solving Using Per-Instance Algorithm Selection". In: *Learning and Intelligent Optimization - 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*. Ed. by Clarisse Dhaenens, Laetitia Jourdan, and Marie-Eléonore Marmion. Vol. 8994. Lecture Notes in Computer Science. Springer, pp. 202–217. DOI: 10.1007/978-3-319-19084-6_18.

Kriegel, Hans-Peter, Peer Kröger, and Arthur Zimek (2009). "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering". In: *TKDD* 3.1, 1:1–1:58. DOI: 10.1145/1497577.1497578.

Kriegel, Hans-Peter et al. (2011). "Density-based clustering". In: *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 1.3, pp. 231–240. DOI: 10.1002/widm.30. URL: https://doi.org/10.1002/widm.30.

Lensen, Andrew, Bing Xue, and Mengjie Zhang (2018). "Automatically evolving difficult benchmark feature selection datasets with genetic programming". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. Ed. by Hernán E. Aguirre and Keiki Takadama. ACM, pp. 458–465. DOI: 10.1145/3205455.3205552.

Li, Bingdong et al. (2016). "Stochastic Ranking Algorithm for Many-Objective Optimization Based on Multiple Indicators". In: *IEEE Trans. Evolutionary Computation* 20.6, pp. 924–938. DOI: 10.1109/TEVC.2016.2549267.

Li, Miqing and Jinhua Zheng (2009). "Spread Assessment for Evolutionary Multi-Objective Optimization". In: *Evolutionary Multi-Criterion Optimization, 5th International Conference, EMO 2009, Nantes, France, April 7-10, 2009. Proceedings*. Ed. by Matthias Ehrgott et al. Vol. 5467. Lecture Notes in Computer Science. Springer, pp. 216–230. DOI: 10.1007/978-3-642-01020-0_20.

Li, Xiang et al. (2011). "Initialization strategies to enhancing the performance of genetic algorithms for the p-median problem". In: *Comput. Ind. Eng.* 61.4, pp. 1024–1034. DOI: 10.1016/j.cie.2011.06.015.

Lloyd, Stuart P. (1982). "Least squares quantization in PCM". In: *IEEE Trans. Inf. Theory* 28.2, pp. 129–136. DOI: 10.1109/TIT.1982.1056489.

López-Ibáñez, Manuel, Luís Paquete, and Thomas Stützle (2010). "Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization". In: *Experimental Methods for the Analysis of Optimization Algorithms*. Ed. by Thomas Bartz-Beielstein et al. Springer, pp. 209–222. DOI: 10.1007/978-3-642-02538-9_9.

López-Ibáñez, Manuel et al. (2016). "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3, pp. 43–58.

Luxburg, Ulrike von (2007). "A tutorial on spectral clustering". In: *Stat. Comput.* 17.4, pp. 395–416. DOI: 10.1007/s11222-007-9033-z. URL: https://doi.org/10.1007/s11222-007-9033-z.

Luxburg, Ulrike von, Robert C. Williamson, and Isabelle Guyon (2012). "Clustering: Science or Art?" In: *Unsupervised and Transfer Learning - Workshop held at ICML 2011, Bellevue, Washington, USA, July 2, 2011*. Ed. by Isabelle Guyon et al. Vol. 27. JMLR Proceedings. JMLR.org, pp. 65–80. URL: http://proceedings.mlr.press/v27/luxburg12a.html.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

Macià, Núria and Ester Bernadó-Mansilla (2014). "Towards UCI+: a mindful repository design". In: *Information Sciences* 261, pp. 237–262. DOI: 10.1016/j.ins.2013.08.059.

Macià, Núria, Albert Orriols-Puig, and Ester Bernadó-Mansilla (2010). "In search of targeted-complexity problems". In: *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11,*

2010. Ed. by Martin Pelikan and Jürgen Branke. ACM, pp. 1055–1062. DOI: 10.1145/1830483.1830674.

Macnaughton-Smith, P. et al. (1964). "Dissimilarity analysis: a new technique of hierarchical sub-division". In: *Nature* 202.4936, p. 1034. DOI: 10.1038/2021034a0.

MacQueen, James (1967). "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, pp. 281–297. URL: https://projecteuclid.org/euclid.bsmsp/1200512992.

Martínez, Saúl Zapotecas, Edgar G. Yáñez Oropeza, and Carlos A. Coello Coello (2011). "Self-adaptation Techniques Applied to Multi-Objective Evolutionary Algorithms". In: *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*. Ed. by Carlos A. Coello Coello. Vol. 6683. Lecture Notes in Computer Science. Springer, pp. 567–581. DOI: 10.1007/978-3-642-25566-3_44.

McDermott, James (2020). "When and Why Metaheuristics Researchers can Ignore "No Free Lunch" Theorems". In: *SN Computer Science* 1.1, p. 60. DOI: 10.1007/s42979-020-0063-3.

McInnes, Leland and John Healy (2018). "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: *CoRR* abs/1802.03426. arXiv: 1802.03426. URL: http://arxiv.org/abs/1802.03426.

McShane, Lisa M. et al. (2002). "Methods for assessing reproducibility of clustering patterns observed in analyses of microarray data". In: *Bioinformatics* 18.11, pp. 1462–1469. DOI: 10.1093/bioinformatics/18.11.1462.

Meilă, Marina (2007). "Comparing clusterings—an information based distance". In: *Journal of multivariate analysis* 98.5, pp. 873–895. DOI: 10.1016/j.jmva.2006.11.013.

Melnykov, Volodymyr and Igor Melnykov (2012). "Initializing the EM algorithm in Gaussian mixture models with an unknown number of components". In: *Comput. Stat. Data Anal.* 56.6, pp. 1381–1395. DOI: 10.1016/j.csda.2011.11.002.

Mersmann, Olaf et al. (2011). "Exploratory Landscape Analysis". In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*.

GECCO '11. Dublin, Ireland: Association for Computing Machinery, pp. 829–836. ISBN: 9781450305570. DOI: 10.1145/2001576.2001690.

Mezura-Montes, Efrén and Carlos A. Coello Coello (2011). "Constraint-handling in nature-inspired numerical optimization: past, present and future". In: *Swarm and Evolutionary Computation* 1.4, pp. 173–194. DOI: 10.1016/j.swevo.2011.10.001.

Michalewicz, Zbigniew and Marc Schoenauer (1996). "Evolutionary algorithms for constrained parameter optimization problems". In: *Evolutionary computation* 4.1, pp. 1–32. DOI: 10.1162/evco.1996.4.1.1.

Michie, Donald, David J. Spiegelhalter, and C. C. Taylor (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood. ISBN: 0-13-106360-X.

Miller, Brad L. and David E. Goldberg (1995). "Genetic Algorithms, Tournament Selection, and the Effects of Noise". In: *Complex Systems* 9.3. URL: http://www.complex-systems.com/abstracts/v09%5C_i03%5C_a02.html.

Morissette, Laurence and Sylvain Chartier (2013). "The k-means clustering technique: General considerations and implementation in Mathematica". In: *Tutorials in Quantitative Methods for Psychology* 9.1, pp. 15–24.

Mühlenbein, Heinz (1992). "How Genetic Algorithms Really Work: Mutation and Hillclimbing". In: *Parallel Problem Solving from Nature 2, PPSN-II, Brussels, Belgium, September 28-30, 1992*. Ed. by Reinhard Männer and Bernard Manderick. Elsevier, pp. 15–26.

Mukhopadhyay, Anirban, Sanghamitra Bandyopadhyay, and Ujjwal Maulik (2006). "Clustering using Multi-objective Genetic Algorithm and its Application to Image Segmentation". In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Taipei, Taiwan, October 8-11, 2006*. IEEE, pp. 2678–2683. DOI: 10.1109/ICSMC.2006.385268.

Mukhopadhyay, Anirban, Ujjwal Maulik, and Sanghamitra Bandyopadhyay (2015). "A Survey of Multiobjective Evolutionary Clustering". In: *ACM Comput. Surv.* 47.4, 61:1–61:46. ISSN: 0360-0300. DOI: 10.1145/2742642.

Mukhopadhyay, Anirban et al. (2014). "Survey of Multiobjective Evolutionary Algorithms for Data Mining: Part II". In: *IEEE Transactions on Evolutionary Computation* 18.1, pp. 25–35. DOI: 10.1109/TEVC.2013.2290082.

Muñoz, Mario A. et al. (2018). "Instance spaces for machine learning classification". In: *Machine Learning* 107.1, pp. 109–147. DOI: 10.1007/s10994-017-5629-5.

Nelder, John A. and R. Mead (1965). "A Simplex Method for Function Minimization". In: *Comput. J.* 7.4, pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

Nemenyi, Peter (1962). "Distribution-free multiple comparisons". In: *Biometrics*. Vol. 18. 2. International Biometric Soc 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, p. 263.

Neshat, Mehdi et al. (2019). "A hybrid evolutionary algorithm framework for optimising power take off and placements of wave energy converters". In: ed. by Anne Auger and Thomas Stützle, pp. 1293–1301. DOI: 10.1145/3321707.3321806.

Oliveira, Carlos et al. (2018). "Mapping the Effectiveness of Automated Test Suite Generation Techniques". In: *IEEE Trans. Reliability* 67.3, pp. 771–785. DOI: 10.1109/TR.2018.2832072.

Ong, Yew Soon et al. (2005). "Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems". In: *Knowledge Incorporation in Evolutionary Computation*. Springer, pp. 307–331.

Ortiz, Miguel León and Ning Xiong (2014). "Investigation of Mutation Strategies in Differential Evolution for Solving Global Optimization Problems". In: *Artificial Intelligence and Soft Computing - 13th International Conference, ICAISC 2014, Zakopane, Poland, June 1-5, 2014, Proceedings, Part I*. Ed. by Leszek Rutkowski et al. Vol. 8467. Lecture Notes in Computer Science. Springer, pp. 372–383. DOI: 10.1007/978-3-319-07173-2_32.

Park, YoungJa and ManSuk Song (1998). "A Genetic Algorithm for Clustering Problems". In: *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Ed. by John R. Koza et al. University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann, pp. 568–575.

Paula Garcia, Rafael de et al. (2017). "A rank-based constraint handling technique for engineering design optimization problems solved by genetic algorithms". In: *Computers & Structures* 187, pp. 77–87.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830. URL: http://dl.acm.org/citation.cfm?id=2078195.

Pei, Yaling and Osmar Zaïane (2006). "A synthetic data generator for clustering and outlier analysis". In: Report. DOI: 10.7939/R3B23S.

Qiu, Weiliang and Harry Joe (2006a). "Generation of Random Clusters with Specified Degree of Separation". In: *J. Classification* 23.2, pp. 315–334. DOI: 10.1007/s00357-006-0018-y.

Qiu, Weiliang and Harry Joe (2006b). "Separation index and partial membership for clustering". In: *Computational statistics & data analysis* 50.3, pp. 585–603. DOI: 10.1016/j.csda.2004.09.009.

Radcliffe, Nicholas J. (1991). "Equivalence Class Analysis of Genetic Algorithms". In: *Complex Systems* 5.2. URL: http://www.complex-systems.com/abstracts/v05%5C_i02%5C_a04.html.

Rasheed, Khaled (1998). "An adaptive penalty approach for constrained genetic-algorithm optimization". In: *Proceedings of the Third Annual Genetic Programming Conference, Morgan Kaufmann, San Francisco, CA*. Citeseer, pp. 584–590.

Rechenberg, Ingo (1978). "Evolutionsstrategien". In: *Simulationsmethoden in der Medizin und Biologie*. Springer, pp. 83–114.

Recht, Benjamin et al. (2019). "Do ImageNet Classifiers Generalize to ImageNet?" In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 5389–5400. URL: http://proceedings.mlr.press/v97/recht19a.html.

Rice, John R. (1976). "The Algorithm Selection Problem". In: vol. 15, pp. 65–118. DOI: 10.1016/S0065-2458(08)60520-3. URL: https://doi.org/10.1016/S0065-2458(08)60520-3.

Rijsbergen, C. J. van (1979). *Information Retrieval*. Butterworth. ISBN: 0-408-70929-4.

Riquelme, Nery, Christian von Lücken, and Benjamín Barán (2015). "Performance metrics in multi-objective optimization". In: *2015 Latin American Computing Conference, CLEI 2015, Arequipa, Peru, October 19-23, 2015*. IEEE, pp. 1–11. DOI: 10.1109/CLEI.2015.7360024.

Rokach, Lior and Oded Maimon (2005). "Clustering Methods". In: *The Data Mining and Knowledge Discovery Handbook*. Ed. by Oded Maimon and Lior Rokach. Springer, pp. 321–352.

Romano, Simone et al. (2014). "Standardized Mutual Information for Clustering Comparisons: One Step Further in Adjustment for Chance". In: *Proceedings of*

the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1143–1151. URL: http://proceedings.mlr.press/v32/romano14.html.

Rothlauf, Franz (2006). *Representations for genetic and evolutionary algorithms (2. ed.)* Springer. ISBN: 978-3-540-25059-3.

Rothlauf, Franz (2011). *Design of Modern Heuristics*. Natural Computing Series. Springer. ISBN: 978-3-540-72961-7. DOI: 10.1007/978-3-540-72962-4.

Rothlauf, Franz and David E. Goldberg (2003). "Redundant Representations in Evolutionary Computation". In: *Evol. Comput.* 11.4, pp. 381–415. DOI: 10.1162/106365603322519288.

Rousseeuw, Peter J. (1987). "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *Journal of computational and applied mathematics* 20, pp. 53–65.

Runarsson, Thomas P. and Xin Yao (2000). "Stochastic ranking for constrained evolutionary optimization". In: *IEEE Transactions on Evolutionary Computation* 4.3, pp. 284–294. ISSN: 1089-778X. DOI: 10.1109/4235.873238.

Runarsson, Thomas P. and Xin Yao (2002). "Continuous selection and self-adaptive evolution strategies". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*. Vol. 1. IEEE, pp. 279–284. DOI: 10.1109/CEC.2002.1006247.

Santana-Quintero, Luis V., Alfredo Arias Montano, and Carlos A. Coello Coello (2010). "A review of techniques for handling expensive functions in evolutionary multi-objective optimization". In: *Computational intelligence in expensive optimization problems*. Springer, pp. 29–59.

Santos, Jorge M. and Mark J. Embrechts (2009). "On the Use of the Adjusted Rand Index as a Metric for Evaluating Supervised Classification". In: *Artificial Neural Networks - ICANN 2009, 19th International Conference, Limassol, Cyprus, September 14-17, 2009, Proceedings, Part II*. Ed. by Cesare Alippi et al. Vol. 5769. Lecture Notes in Computer Science. Springer, pp. 175–184. DOI: 10.1007/978-3-642-04277-5_18.

Sastry, Kumara and David E. Goldberg (2001). "Modeling tournament selection with replacement using apparent added noise". In: *Intelligent Engineering Systems Through Artificial Neural Networks, 11, 129–134.(Also IlliGAL*. Citeseer.

Schaffer, J. D. and L. J. Eshelman (1991). "On Crossover as an Evolutionary Viable Strategy". In: *Proceedings of the 4th International Conference on Genetic Algorithms*. Ed. by R.K. Belew and L.B. Booker. Morgan Kaufmann, pp. 61–68.

Schmidhuber, Jürgen (2015). "Deep learning in neural networks: An overview". In: *Neural Networks* 61, pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

Schraudolph, Nicol N. and Richard K. Belew (1992). "Dynamic Parameter Encoding for Genetic Algorithms". In: *Mach. Learn.* 9, pp. 9–21. DOI: 10.1007/BF00993252.

Schwefel, Hans-Paul (1977). "Evolutionsstrategien für die numerische Optimierung". In: *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Springer, pp. 123–176.

Schweizer, B. and A. Sklar (1960). "Statistical metric spaces." In: *Pacific Journal of Mathematics* 10.1, pp. 313–334. URL: https://projecteuclid.org:443/euclid.pjm/1103038644.

Shaefer, Craig G. (1987). "The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique". In: *Proceedings of the 2nd International Conference on Genetic Algorithms, Cambridge, MA, USA, July 1987*. Cambridge, Massachusetts, USA: Lawrence Erlbaum Associates, pp. 50–58. ISBN: 0-8058-0158-8. URL: http://dl.acm.org/citation.cfm?id=42512.42520.

Shand, Cameron et al. (2018). "Towards an adaptive encoding for evolutionary data clustering". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. ACM, pp. 521–528. DOI: 10.1145/3205455.3205506.

Shand, Cameron et al. (2019). "Evolving controllably difficult datasets for clustering". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. ACM, pp. 463–471. DOI: 10.1145/3321707.3321761.

Sibson, Robin (1973). "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method". In: *Comput. J.* 16.1, pp. 30–34. DOI: 10.1093/comjnl/16.1.30.

Smith-Miles, Kate (2008). "Cross-disciplinary perspectives on meta-learning for algorithm selection". In: *ACM Comput. Surv.* 41.1, 6:1–6:25. DOI: 10.1145/1456650.1456656.

Smith-Miles, Kate and Simon Bowly (2015). "Generating new test instances by evolving in instance space". In: *Computers & Operations Research* 63, pp. 102–113.

Smith-Miles, Kate and Leo Lopes (2012). "Measuring instance difficulty for combinatorial optimization problems". In: *Computers & Operations Research* 39.5, pp. 875–889. DOI: 10.1016/j.cor.2011.07.006.

Smith-Miles, Kate and Thomas T. Tan (2012). "Measuring algorithm footprints in instance space". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*. IEEE, pp. 1–8. DOI: 10.1109/CEC.2012.6252992.

Smith-Miles, Kate et al. (2014). "Towards objective measures of algorithm performance across instance space". In: *Computers & Operations Research* 45, pp. 12–24. DOI: 10.1016/j.cor.2013.11.015.

Soares, Rodrigo G. F., Teresa Bernarda Ludermir, and Francisco de A. T. de Carvalho (2009). "An Analysis of Meta-learning Techniques for Ranking Clustering Algorithms Applied to Artificial Data". In: *Artificial Neural Networks - ICANN 2009, 19th International Conference, Limassol, Cyprus, September 14-17, 2009, Proceedings, Part I*. Ed. by Cesare Alippi et al. Vol. 5768. Lecture Notes in Computer Science. Springer, pp. 131–140. DOI: 10.1007/978-3-642-04274-4_14.

Steinley, Douglas (2004). "Properties of the Hubert-Arable Adjusted Rand Index." In: *Psychological methods* 9.3, p. 386. DOI: 10.1037/1082-989X.9.3.386.

Stewart, G. W. (1980). "The Efficient Generation of Random Orthogonal Matrices with an Application to Condition Estimators". In: *SIAM Journal on Numerical Analysis* 17.3, pp. 403–409. ISSN: 00361429. URL: http://www.jstor.org/stable/2156882.

Surry, Patrick D. and Nicholas J. Radcliffe (1996). "Formal Algorithms + Formal Representations = Search Strategies". In: *Parallel Problem Solving from Nature - PPSN IV, International Conference on Evolutionary Computation. The 4th International Conference on Parallel Problem Solving from Nature, Berlin, Germany, September 22-26, 1996, Proceedings*. Ed. by Hans-Michael Voigt et al. Vol. 1141. Lecture Notes in Computer Science. Springer, pp. 366–375. DOI: 10.1007/3-540-61723-X_1001.

Syswerda, Gilbert (1989). "Uniform Crossover in Genetic Algorithms". In: *Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989*. Morgan Kaufmann, pp. 2–9. URL: http://dl.acm.org/citation.cfm?id=645512.657265.

Talbi, El-Ghazali (2009). *Metaheuristics - From Design to Implementation*. Wiley. ISBN: 978-0-470-27858-1.

Tibshirani, Robert, Guenther Walther, and Trevor Hastie (2001). "Estimating the number of clusters in a data set via the gap statistic". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2, pp. 411–423.

Trautmann, Heike et al. (2008). "A Convergence Criterion for Multiobjective Evolutionary Algorithms Based on Systematic Statistical Testing". In: *International Conference on Parallel Problem Solving from Nature*. Springer, pp. 825–836. DOI: 10.1007/978-3-540-87700-4_82.

Trautmann, Heike et al. (2009). "Statistical Methods for Convergence Detection of Multi-Objective Evolutionary Algorithms". In: *Evolutionary Computation* 17.4, pp. 493–509. DOI: 10.1162/evco.2009.17.4.17403.

Vanschoren, Joaquin (2019). "Meta-Learning". In: *Automated Machine Learning - Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. The Springer Series on Challenges in Machine Learning. Springer, pp. 35–61. DOI: 10.1007/978-3-030-05318-5_2.

Varmuza, Kurt and Peter Filzmoser (2009). *Introduction to multivariate statistical analysis in chemometrics*. CRC Press. ISBN: 9781420059472.

Vendramin, Lucas, Ricardo J. G. B. Campello, and Eduardo R. Hruschka (2010). "Relative clustering validity criteria: A comparative overview". In: *Statistical Analysis and Data Mining* 3.4, pp. 209–235. DOI: 10.1002/sam.10080.

Virtanen, Pauli et al. (2019). "SciPy 1.0-Fundamental Algorithms for Scientific Computing in Python". In: *CoRR* abs/1907.10121. arXiv: 1907.10121. URL: http://arxiv.org/abs/1907.10121.

Walter, Bruce et al. (2008). "Fast agglomerative clustering for rendering". In: *2008 IEEE Symposium on Interactive Ray Tracing*. IEEE, pp. 81–86.

Ward Jr, Joe H. (1963). "Hierarchical grouping to optimize an objective function". In: *Journal of the American statistical association* 58.301, pp. 236–244.

Whitley, L. Darrell (1989). "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best". In: *Proceedings of*

the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989. Ed. by J. David Schaffer. Morgan Kaufmann, pp. 116–123.

Whitley, L. Darrell, Keith E. Mathias, and Patrick A. Fitzhorn (1991). "Delta Coding: An Iterative Search Strategy for Genetic Algorithms". In: *Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991*. Ed. by Richard K. Belew and Lashon B. Booker. Morgan Kaufmann, pp. 77–84.

Wolpert, David H. and William G. Macready (1997). "No free lunch theorems for optimization". In: *IEEE Trans. Evolutionary Computation* 1.1, pp. 67–82. DOI: 10.1109/4235.585893.

Yang, Bo et al. (2017). "Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 3861–3870. URL: http://proceedings.mlr.press/v70/yang17b.html.

Yeniay, Özgür (2005). "Penalty function methods for constrained optimization with genetic algorithms". In: *Mathematical and computational Applications* 10.1, pp. 45–56. DOI: 10.3390/mca10010045.

Yim, Odilia and Kylee T. Ramdeen (2015). "Hierarchical cluster analysis: comparison of three linkage measures and application to psychological data". In: *The quantitative methods for psychology* 11.1, pp. 8–21.

Zaït, Mohamed and Hammou Messatfa (1997). "A comparative study of clustering methods". In: *Future Gener. Comput. Syst.* 13.2-3, pp. 149–159. DOI: 10.1016/S0167-739X(97)00018-6.

Zhu, Shuwei, Lihong Xu, and Erik D. Goodman (2020). "Evolutionary multi-objective automatic clustering enhanced with quality metrics and ensemble strategy". In: *Knowl. Based Syst.* 188. DOI: 10.1016/j.knosys.2019.105018.

Zitzler, Eckart, Joshua D. Knowles, and Lothar Thiele (2008). "Quality Assessment of Pareto Set Approximations". In: *Multiobjective Optimization, Interactive and Evolutionary Approaches [outcome of Dagstuhl seminars]*. Ed. by Jürgen Branke et al. Vol. 5252. Lecture Notes in Computer Science. Springer, pp. 373–404. DOI: 10.1007/978-3-540-88908-3_14.

Zitzler, Eckart and Lothar Thiele (1998). "Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study". In: *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*. Ed. by A. E. Eiben et al. Vol. 1498. Lecture Notes in Computer Science. Springer, pp. 292–304. DOI: 10.1007/BFb0056872.

Zitzler, Eckart et al. (2003). "Performance assessment of multiobjective optimizers: an analysis and review". In: *IEEE Trans. Evolutionary Computation* 7.2, pp. 117–132. DOI: 10.1109/TEVC.2003.810758.

Appendix A

HAWKS Supplementary Material

A.1 HAWKS availability

HAWKS is available as a Python package on PyPI at https://pypi.org/project/hawks/, or on GitHub at https://github.com/sea-shunned/hawks. Instructions for installation and documentation on how to the use the package can be found there.

A.1.1 Replicating our experiments

For each experiment that involves HAWKS in this thesis, we have provided a link to a configuration file (in the text) for reproducibility. In the GitHub repository that contains these configuration files (https://github.com/sea-shunned/thesis_material/), the version of HAWKS used in this thesis (v1.0) is also included to ensure these files work regardless of future changes/development to HAWKS. There is also guidance on how to use these configuration files and generate the graphs included in this thesis.

A.2 Random sampling sensitivity

To ascertain whether different samplings (or *instantiations*) of the same genotype in HAWKS results in significant variance in the phenotype, a sensitivity analysis was performed. For this, a single genotype is selected from the final population and sampled 50 times using different seeds. The magnitude of this observed variance is not informative in isolation, however, so the variance in the fitness of the

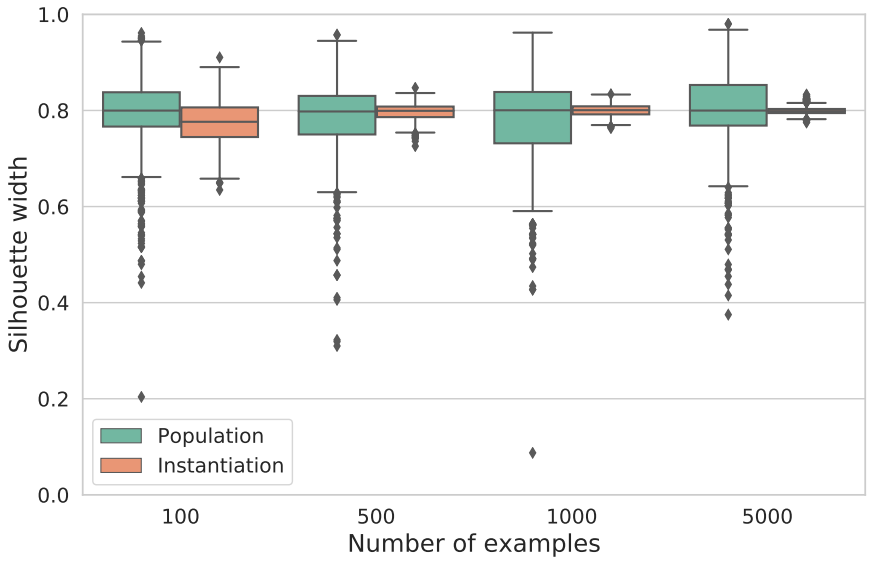


Figure A.1: Sensitivity analysis of the silhouette width from different instantiations of the same genotype relative to the silhouette width variance across the whole population, for an increasing number of data points (where $K = 5$).

final population itself is also recorded (for parity, the population size is 50). The number of examples was increased $N \in \{100, 500, 1000, 5000\}$ with a constant $K = 5$ clusters. To ensure diversity among the datasets in the population, the variance in the silhouette width for multiple instantiations of a genotype should be less than the variance of fitness across a population, so that we are obtaining different cluster structures rather than just different samples from the same distribution of clusters.

In Figure A.1 we can see that, as the number of examples increases, the variance in the silhouette width of different instantiations decreases dramatically, particularly relative to the variance across the population. With the aforementioned sensitivity of the silhouette width, it is expected that as N increases the sampling density of the clusters increases such that different samplings are mostly equivalent (in terms of average distance to other points) i.e. the density is sufficient to overcome differences in sampling locations. This does indicate that when generating very small datasets (i.e. < 100 examples) or few data points per cluster, HAWKS will have greater sensitivity to the particular instantiations for that run. Thus, the N and K parameters need to be considered together, as it is their combination that influence the sampling density and thus sensitivity to instantiation.

A.3 Generating random cluster sizes

To generate K randomly sized clusters that sum to a given value, and have at least a certain size, the following approach is used. First, we generate K weights:

$$\mathbf{w} = [w_1, \ldots, w_K]$$

where $w_i \sim \text{Dir}(\alpha)$ and $\alpha = (1, 1, \ldots, 1)$. For a given cluster i, the size is:

$$|C_i| = C_{\min} + w_i(N - (K \times C_{\min}))$$

where $C_{\min}$ is the minimum size a cluster can be, and N is the number of data points.

This ensures a uniform distribution of cluster sizes *after* scaling such that they sum to N, which is not guaranteed when simply sampling random numbers and scaling these to N (as the resulting distribution is not uniform).

A.4 Modified tournament selection

In order to limit the introduction of preference towards either the fitness or the constraints outside of setting a value for the P_f, we modify the well-established binary tournament selection protocol for the parental selection step of a GA. As discussed in Section 3.2.2, a subset of individuals are selected for a tournament, from which the individual with the highest fitness is selected as a parent. Typically, a binary tournament is used, so only two random individuals are selected. This adds selection pressure from the fitness, which may run counter to the preference specified by the stochastic ranking parameter P_f, particularly if it is set < 0.5 such that the constraints are favoured. Our modification is thus that, still using a binary tournament, the individual with the higher rank in the sorted (by stochastic ranking) population is used.

To investigate how this modified the characteristics of the algorithm, we run a simple experiment. The experimental set up is similar to Section 5.2.3, i.e. only the *overlap* constraint is used, and we use $P_f \in \{0.1, 0.5, 0.9\}$ to explore how the addition of preference with the parental selection changes with the preference embedded by P_f. As before, we run HAWKS 30 times at 2D and 20D, with s_{target} 0.6 and 0.2 respectively.

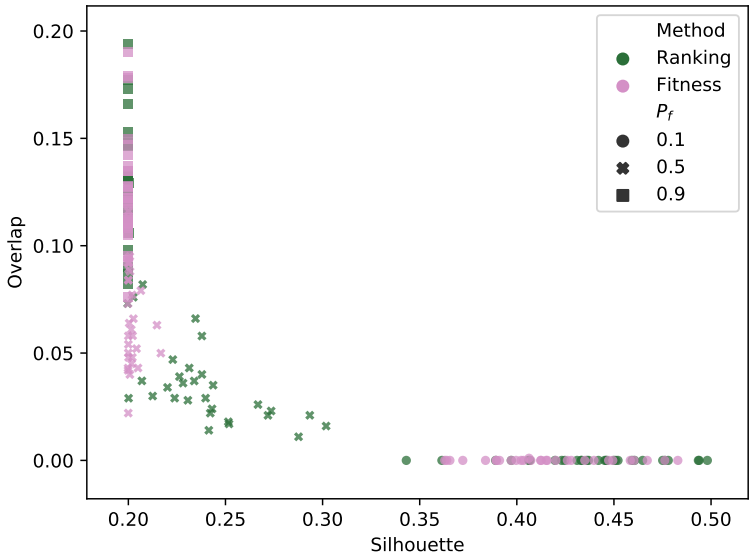


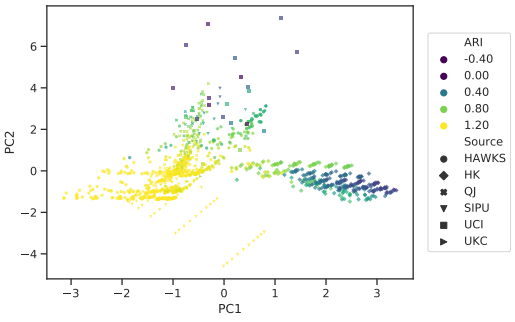
Figure A.2: Silhouette width vs. *overlap* for different values of P_f in 20D, using the original and modified binary tournament.

From our experiment in Section 5.2.3 we know that the trade-off between the objective and constraint occurred in 20D and less so in 2D, where they could both be satisfied. Thus, here in Figure A.2 (similar to Figure 5.8b) we show the silhouette width against the *overlap* in 20D. When using the original tournament selection method, the additional pressure towards optimality (over constraint satisfaction) has reduced the level of trade-off between the objective and *overlap* constraint.

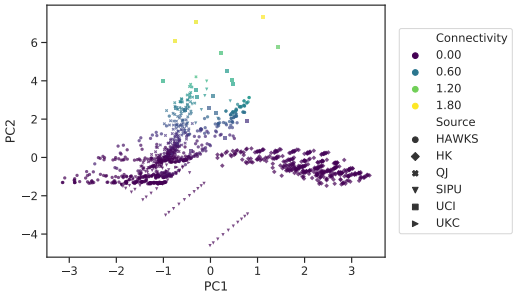
A.5 Exploring the instance space additional material

A.5.1 Expanded instance space features

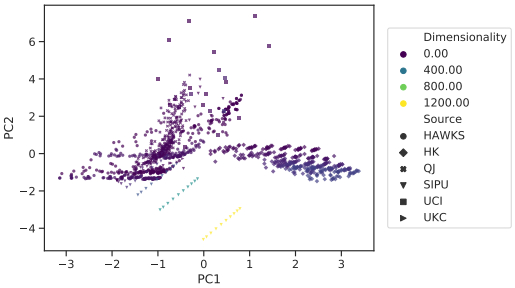
Figure A.3 shows the instance space from Section 6.2 colourized by all problem features.



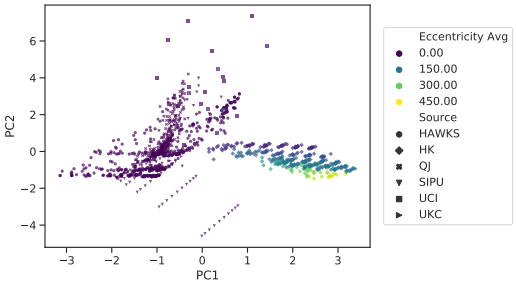
(a) ARI



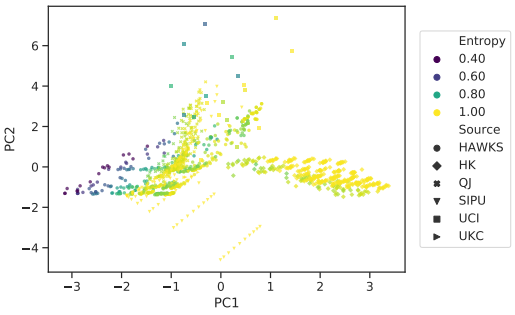
(b) Connectivity



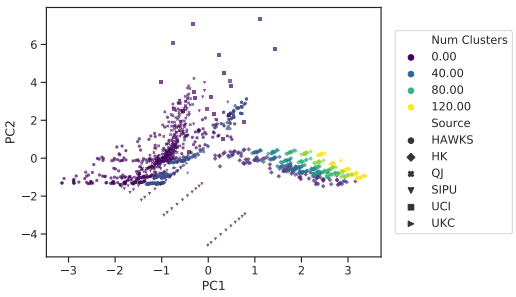
(c) Number of dimensions



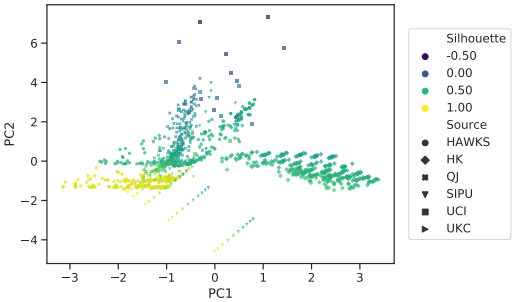
(d) Average eccentricity



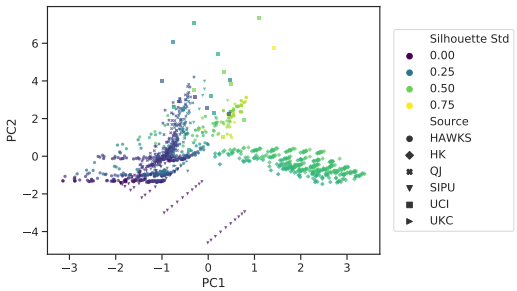
(e) Entropy of cluster sizes



(f) Number of clusters



(g) Silhouette width



(h) Standard deviation of silhouette width

Figure A.3: All problem feature values across the instance space

Appendix B

MOCK Supplementary Material

B.1 Instance space

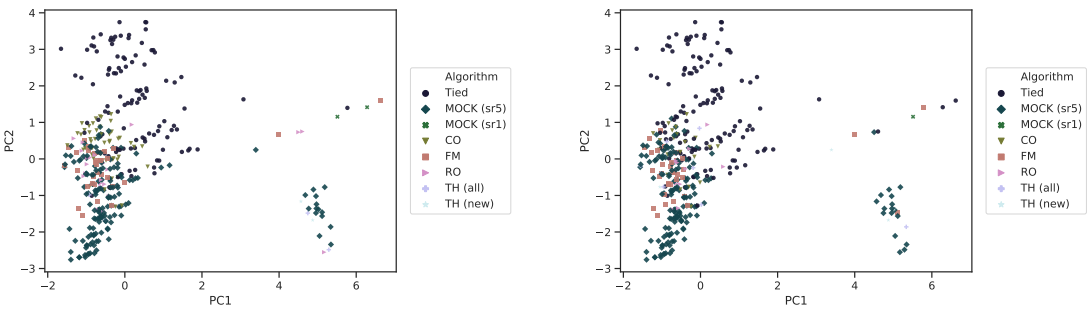
In this section we provide graphs that further illustrate and describe the instance space presented in Section 7.3.2.

B.1.1 Winner selection method

Figure 7.13 showed the 'winning' algorithm, for which the *median* value of the best individuals from each of the 30 runs was used for each dataset. For completeness, in Figure B.1 we show this side-by-side with comparing the *maximum* ARI achieved across the 30 runs for each method, to see if the selection method resulted in different results. The results are also tabulated in Table B.1. The main difference that can be observed is the increased number of 'tied' results, which is expected as the performance on these datasets is generally high enough that most approaches could do equally well given enough runs. In particular, the *CO* method was primarily affected by using the maximum instead, likely due to the search strategies favouring different datasets in different ways through either a bias towards the connectivity objective by *RO*, or towards the intra-cluster variance objective by the strategies that raise the mutation rate, and the more neutral behaviour of *CO*.

B.1.2 Method performance across the space

In Figure 7.12 we showed the instance space for the HAWKS datasets generated in Section 6.2.5, the *UKC*, and the (expanded) *HK* datasets. In Figure B.1 we



(a) Median (b) Maximum

Figure B.1: The instance space of the datasets highlighted according to the method that achieved the best performance as measured by the median, (a), or the max, (b), ARI of the best individuals from the 30 runs.

Table B.1: Number of 'wins' for each approach across the 378 datasets, for the two individual selection methods (using either the median or maximum ARI of the best individuals from the 30 runs).

	Number of 'wins'	
Method	Median	Maximum
Δ-MOCK ($sr5$)	155	160
Tied	109	136
CO	47	34
FM	33	22
RO	24	11
TH_{all}	4	8
TH_{new}	3	6
Δ-MOCK ($sr1$)	3	1

can see these datasets according to the 'best' approach. In Figure B.2 we show the actual ARI obtained on each of the datasets for the different methods. The consistent gradient of performance across the space is particularly encouraging for trying to address the ASP, i.e. predicting the performance according to the problem features. If there was no correlation between the projection of the datasets and the performance then it would indicate that the problem features do not reflect the difficulty for these methods.

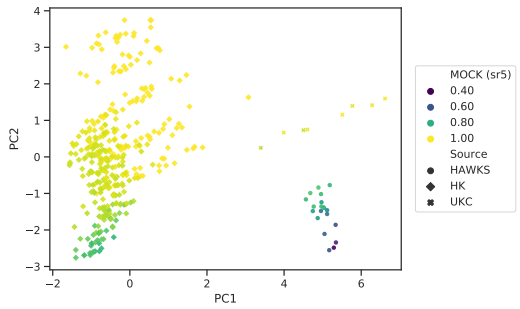
B.1.3 Full problem feature values

For a fuller categorization of how the problem features vary across the instance space, in Figure B.3 we show the instance space according to each of the problem features.

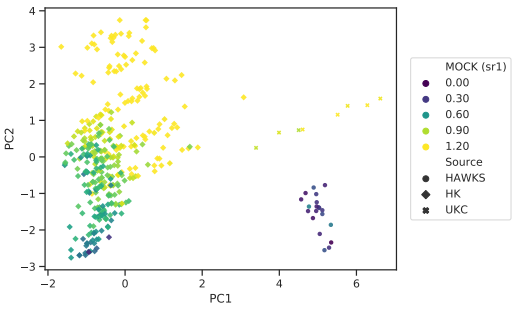
B.1.4 Expanded δ ranges

In this section, we provide the results for running a wider range of δ values on the HAWKS datasets. Based on the particularly poor performance for Δ-MOCK ($sr1$), it is clear that even our 'non-restrictive' δ value may have been too high. Therefore, it follows to use different settings for δ_{Low} and δ_{High} to further understand if these datasets have some inherent difficulty for Δ-MOCK that led to poor performance, or if the search space was just too restricted. For this, we shift the values such that $\delta_{\text{Low}} = sr9$ and $\delta_{\text{High}} = sr5$, so that our previous larger search space is now our starting point. This enables us to see if there is additional utility in opening up the search space further.

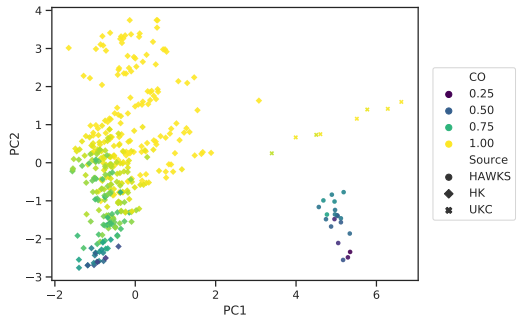
In Figure B.4 we can see the performance and computation times for the search strategies across the three trigger mechanisms. The higher resolution ($sr9$) is statistically significantly better than the previous ($sr5$), indicating that having a larger search space is beneficial for these datasets. Notably, however, we see a vast difference in performance for the search strategies across the trigger mechanisms. Both from the boxplots and the mean ARI values (Table B.2), in absolute terms the ARI values obtained are quite similar, particularly when compared to the previous experiment. Thus, before we discuss the significant differences between the methods it is clear that having a non-restrictive value of δ is more important than a particular strategy, suggesting that if the optimal solution(s) are unable to be found then no strategy can aid with this.



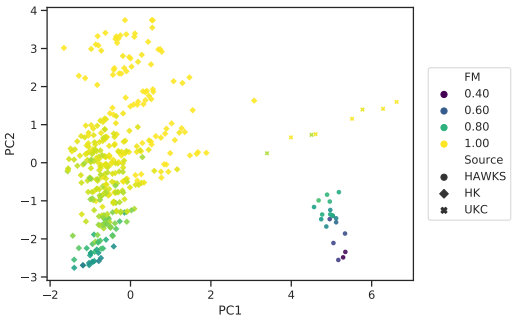
(a) Δ-MOCK *sr*5



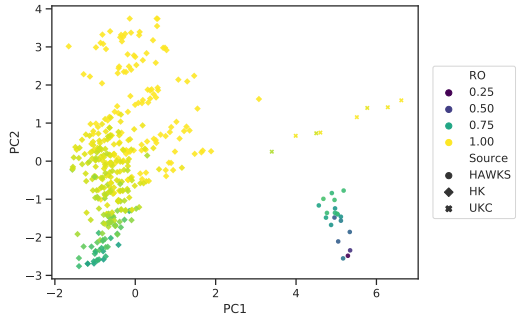
(b) Δ-MOCK *sr*1



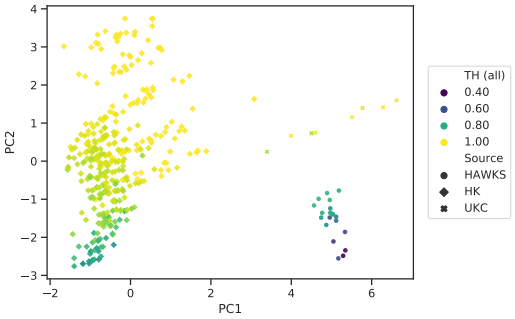
(c) *CO*



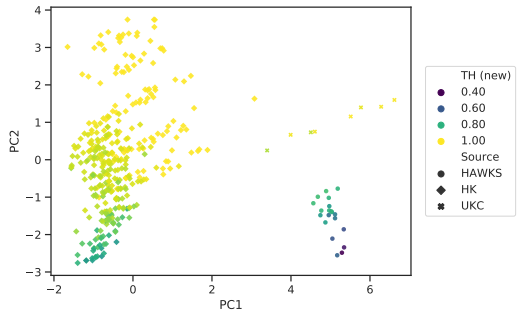
(d) *FM*



(e) *RO*

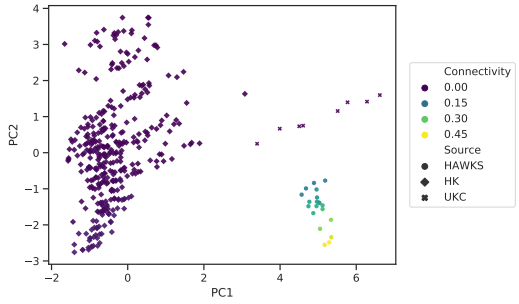


(f) TH_{all}

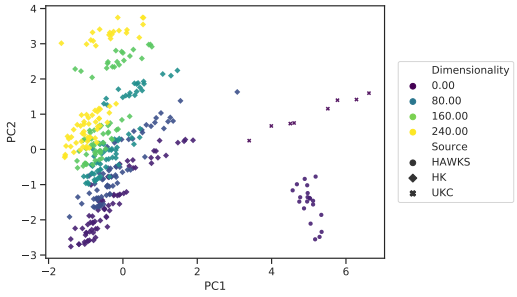


(g) TH_{new}

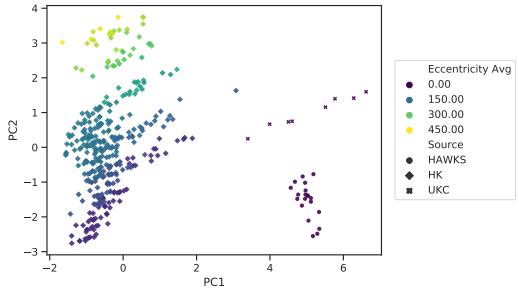
Figure B.2: Performance (ARI) across the instance space for each of the two baseline methods and five search strategies.



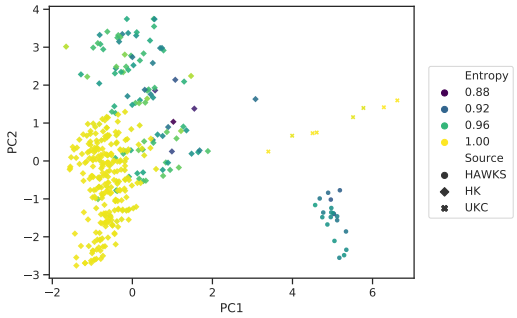
(a) Connectivity



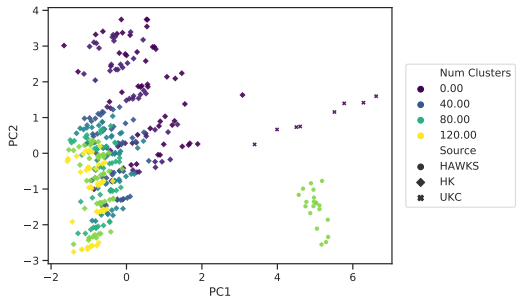
(b) Number of dimensions



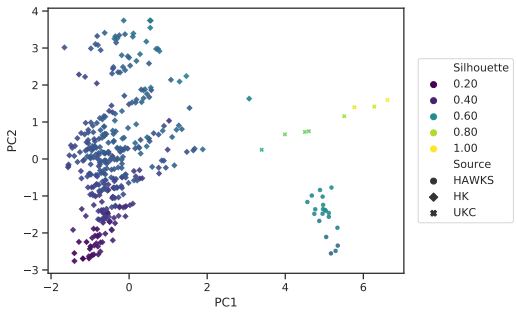
(c) Average eccentricity



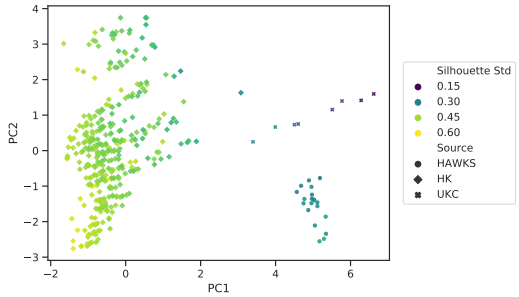
(d) Entropy of cluster sizes



(e) Number of clusters

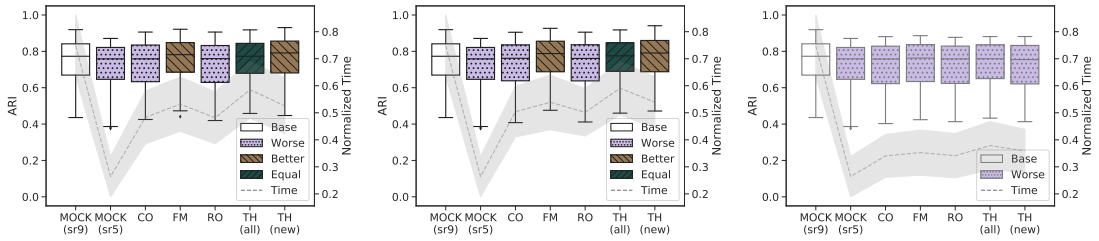


(f) Silhouette width



(g) Standard deviation of silhouette width

Figure B.3: All problem feature values across the instance space



(a) *Random* trigger (b) *Interval* trigger (c) *HV*-based trigger

Figure B.4: Performance for each search strategy for each of the three trigger mechanisms for the HAWKS datasets. The computation times are normalized in the range $[0, 1]$.

Table B.2: ARI mean and standard deviation for HAWKS datasets with $\delta_{\text{Low}} = sr9$ & $\delta_{\text{High}} = sr5$

Method	*Random*	*Interval*	*HV*
CO	0.733 ± 0.121	0.733 ± 0.119	0.723 ± 0.121
FM	0.761 ± 0.110	0.764 ± 0.110	0.731 ± 0.117
RO	0.730 ± 0.123	0.733 ± 0.119	0.723 ± 0.118
TH_{all}	0.751 ± 0.112	0.753 ± 0.110	0.733 ± 0.115
TH_{new}	0.760 ± 0.116	0.764 ± 0.116	0.725 ± 0.120
Δ-MOCK ($sr9$)		0.751 ± 0.113	
Δ-MOCK ($sr5$)		0.720 ± 0.116	

All search strategies performed worse with the *HV* trigger mechanism, indicating that it was too conservative with activating. For this experiment, there were an average of 1.37 triggers (compared to 3.24 on these datasets when the initial δ value was higher). With the wider initial search space, the initial reference gradient is shallow enough that further improvements are unlikely to be 'stagnating' in comparison, creating a situation where it is difficult to trigger further. Thus, this approach only works when the initial δ is considerably restrictive, which in this case it is not. Thus, the additional increases in the search space for the other mechanisms create enough of a difference that the an extra ≈ 0.03 ARI is obtained, making a statistically significant difference.

For the search strategies themselves, we can see that *FM* and TH_{new} were significantly better than Δ-MOCK (*sr*9) for the *random* and *interval* trigger mechanisms. The increased mutation rate is obviously beneficial to increase K as we previously discussed, but of interest is how *FM* was more successful than TH_{all}. It is likely that the latter approach is too disruptive, a property that becomes more important as the genotype length grows and the effect of the hypermutation rate on increasing K is more pronounced. This is supported by the number of clusters in the final population, which is shown in Figure B.5 (for the *interval* mechanism). The poor performance of *RO* is a consequence (once again) of the underestimation of K, as the initialization routine becomes sparser in its sampling of individuals from the MST using different K values as the *true* K increases.

With the average K found by the methods being noticeably lower than the true K, even with the higher δ, it indicates that some other hyperparameters (such as the neighbourhood parameter L) may also be unsuitable for these datasets. Considering that the parameters were selected in the original work primarily for the *HK* datasets, this is unsurprising. As with δ, tuning of such parameters is an important part of Δ-MOCK yet is not practical (particularly for larger datasets).

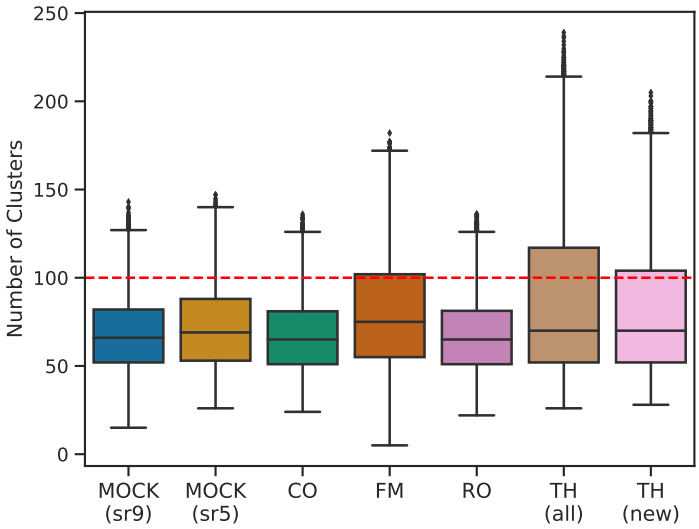


Figure B.5: The number of clusters encoded by individuals in the final population for each method aggregated across all 20 HAWKS datasets, using $\delta_{\text{Low}} = sr9$ and $\delta_{\text{High}} = sr5$. The true number of clusters is shown by the dashed horizontal line.