Evolving Controllably Difficult Datasets for Clustering

Cameron Shand
School of Computer Science,
University of Manchester
cameron.shand@manchester.ac.uk

Richard Allmendinger
Alliance Manchester Business School,
University of Manchester

Julia Handl
Alliance Manchester Business School,
University of Manchester

Andrew Webb
School of Computer Science,
University of Manchester

John Keane
School of Computer Science,
University of Manchester

ABSTRACT

Synthetic datasets play an important role in evaluating clustering algorithms, as they can help shed light on consistent biases, strengths, and weaknesses of particular techniques, thereby supporting sound conclusions. Despite this, there is a surprisingly small set of established clustering benchmark data, and many of these are currently handcrafted. Even then, their difficulty is typically not quantified or considered, limiting the ability to interpret algorithmic performance on these datasets. Here, we introduce HAWKS, a new data generator that uses an evolutionary algorithm to evolve cluster structure of a synthetic data set. We demonstrate how such an approach can be used to produce datasets of a pre-specified difficulty, to trade off different aspects of problem difficulty, and how these interventions directly translate into changes in the clustering performance of established algorithms.

CCS CONCEPTS

• **Theory of computation** → *Design and analysis of algorithms*; Continuous optimization; • **Computing methodologies** → Cluster analysis.

KEYWORDS

Clustering test problems; problem generator; evolutionary optimization; benchmarking; test suite

ACM Reference Format:
Cameron Shand, Richard Allmendinger, Julia Handl, Andrew Webb, and John Keane. 2019. Evolving Controllably Difficult Datasets for Clustering. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321761

1 INTRODUCTION

The literature on evolutionary computation and machine learning offers a varied choice of algorithms. As the no free lunch theorems for optimization and learning suggest that no single algorithm will be superior for every task [27, 28], the practical choice of an effective algorithm for a given instance needs to be informed by an understanding of problem properties as well as the suitability of any particular algorithm with respect to those properties. Benchmark sets play an instrumental role in providing this kind of insight, as they facilitate thorough empirical comparison that can help reveal particular algorithms' areas of strength and weakness.

There is a long tradition of using real-world benchmark sets to assist algorithm comparison in the machine learning and optimization communities. Unfortunately, for some problems, publicly available data can be scarce and is often poorly understood. However, without a sound understanding of problem properties that are of interest, it can be challenging to compile a collection of real-world datasets that provide a comprehensive and suitably challenging coverage of the problem space. Analysis by Macià and Bernadó-Mansilla [12] found a surprising similarity in complexity across the datasets in the UCI Machine Learning Repository [3], which underlines the difficulty of identifying a diverse set of problem instances without an explicit effort to represent particular aspects of problem structure. To create useful benchmarks and empirically test algorithms in an insightful way, an understanding of the core factors that contribute to difficulty is paramount [9, 23].

Given the above, synthetic benchmarks are often preferred over real-world benchmarks as they offer an opportunity to explicitly model problem properties of interest and exercise direct control over these properties [22]. This is the case in both the machine learning and optimization communities. In-depth understanding of the construction and properties of such synthetic benchmarks can be particularly helpful in interpreting algorithm results and in pinpointing systematic variations in algorithm performance (where these arise as a result of the controllable problem properties).

One of the core criticisms leveled against synthetic benchmarks is that they may not be sufficiently complex to capture the properties of real-world data. There is a concern that results on such simplified synthetic benchmarks risk being empirically misleading and may limit practical applicability of algorithms whose development overly relies on such data [9]. As a result, it is generally considered good practice to take a dual approach to algorithm development involving evaluation on both synthetic and real-world benchmarks, leveraging the strengths of both approaches. Arguably, as explored in the work of Smith-Miles [22], the ability of synthetic generators to support the exploration and understanding of the available instance space can, indirectly, also support appropriate design of the composition of real-world benchmark sets.

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321761

Compiling meaningful benchmark sets is particularly challenging for the task of cluster analysis, due to the unsupervised nature of the problem. For real-world data, the generating mechanism is typically unknown and the accuracy of class labels (where available) may be disputed. In the absence of knowledge regarding the optimal clustering solution, assessing performance relies on the evaluation of internal measures of cluster validity. These measures use information intrinsic to the data to assess the prominence of cluster structure. Previous work [7] has argued that such measures (known as cluster validity indices) broadly consider a combination of measures relating to compactness, spatial separation and (to a lesser extent), connectedness of the clusters.

As most clustering algorithms work by considering these same aspects of cluster structure (usually focusing on one of the above categories only), this type of internal assessment easily introduces a bias towards particular algorithms. In particular, it may severely impact the assessment of an algorithm's performance when the assumptions of the validation index are inconsistent with the algorithm's assumptions [17]. This issue can be reduced but not fully addressed through the use of a set of carefully selected (complementary) internal validity measures.

For this reason, synthetic data plays a prominent role in the evaluation of clustering performance. These datasets offer knowledge of the generation mechanism, allowing the use of external indices, which require knowledge of the correct cluster labels but provide an unbiased assessment of clustering performance. Understanding the generating mechanisms of the data also means that interactions between generating mechanism, clustering algorithms, and internal validity indices can be explicitly taken into account and explored. Despite these advantages of synthetic clustering benchmarks, there have been few attempts to develop cluster generators that can systematically vary aspects of problem difficulty. Here, we take inspiration from previous work demonstrating the use of evolutionary computation to generate benchmark problems that provide flexibility, permitting the tuning of parameters to generate a diverse range of problem instances [11]. Such generators have been described for a range of applications, ranging from software testing [5] to combinatorial optimization [25] to genetics [8].

Our paper introduces a synthetic data generator, HAWKS, that uses an evolutionary algorithm (EA) to evolve a population of datasets, optimizing to a specified value of the silhouette width (an internal cluster validity index). We describe in Section 3 the design choices for the underlying EA, and show how some of our generator's key parameters interact to influence the complexity of the resulting dataset. The capability of the generator is demonstrated in Section 4 by highlighting observable changes in the performance of clustering methods across an increasingly complex suite of data sets, and by contrasting this to the clustering performance observed for datasets from other generators. As the tool is designed to be easy-to-use, we hope that it is useful for anything from quick, simple data generation to a creation of a more comprehensive test suite for thorough empirical comparison. Finally, we conclude and suggest future work in Section 5.

2 RELATED WORK

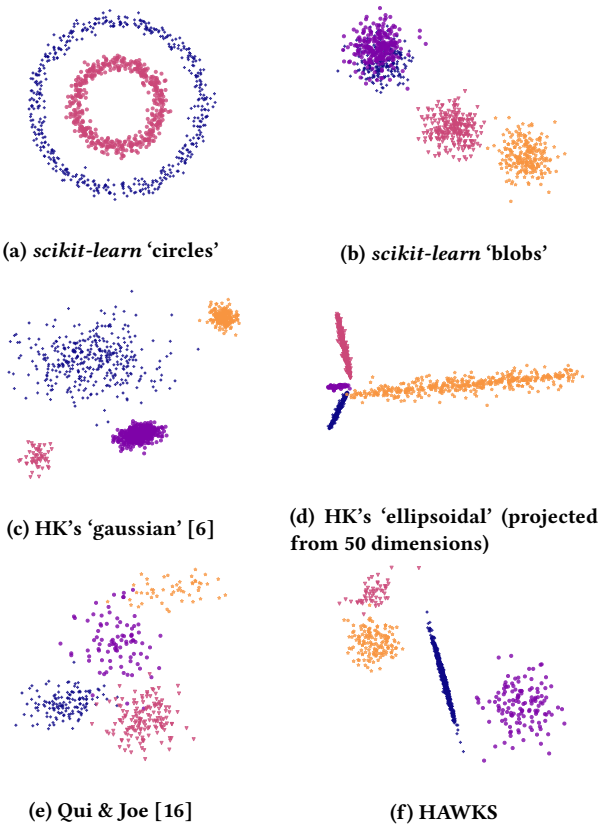
The use of genetic algorithms to evolve datasets has been explored extensively in the works of Smith-Miles, building upon a framework proposed by Rice [14, 18, 21, 22]. The central idea is the creation of an "instance space"—a set of properties that sufficiently describe a dataset such that the datasets themselves can be visualised in this space. This facilitates both the identification of sub-spaces in which a particular algorithm performs well, as well as areas in this space where current benchmarks/suites do not provide any datasets.

Generating a meaningful instance space using cluster validity indices presents an interesting challenge, as a set of non-correlating indices that sufficiently capture the possible properties of a set of clusters would need to be identified. The difficulty with this is highlighted by the extensive comparison of cluster validity indices performed by Arbelaitz et al. [1], which systematically compared 30 indices and found that, although some indicators were redundant, many were suitable to particular situations and thereby added value in their own right. Creating a comprehensive instance space for clustering (using many different cluster validity indices) will be of interest for maximizing the diversity of instances obtainable with our tool (and rigorously assessing the diversity of other collections of clustering benchmarks). Here we limit ourselves to a proof-of-principle, showing how HAWKS can create a simple instance space using a few straightforward features and how this can already start to shed light on the diversity of benchmark data through tangible changes in the performance of clustering algorithms.

Existing data generators for clustering are either naïve, without much consideration of creating an interesting cluster structure, or rely on extensive hand-crafting of certain data properties. Figure 1 illustrates this with some examples of data generators from *scikit-learn*, a popular Python machine learning framework [15], the popular generator from Handl & Knowles (henceforth HK) [6], and Qui & Joe's generator (henceforth QJ) [16].

The generators in *scikit-learn* are useful for quick random generation of structured data, but are either overly contrived (see Figure 1a) or do not provide enough control (Figure 1b) to create benchmarks that are sufficiently challenging to tease out systematic differences between clustering methods.

HK's generator has been used extensively in the literature, and consists of two separate components: 'gaussian' and 'ellipsoidal'. The first of these 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 — an example data set is shown in Figure 1c. The ellipsoidal generator uses a genetic algorithm to shift cluster positions but 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 a measure of compactness while penalizing overlap. An example of a resulting dataset is shown in Figure 1d. Given the above, and (additionally) 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 their ability to generate benchmarks that systematically test the performance of clustering methods with respect to particular aspects of problem difficulty.



(a) *scikit-learn* 'circles'

(b) *scikit-learn* 'blobs'

(c) HK's 'gaussian' [6]

(d) HK's 'ellipsoidal' (projected from 50 dimensions)

(e) Qui & Joe [16]

(f) HAWKS

Figure 1: Representative example datasets produced by different cluster generators. As shown in (f), HAWKS can produce datasets with clusters of very different eccentricities.

The generator described in [16] uses a geometric framework for cluster placement, which is designed to consider a single aspect of cluster quality (the degree of separation between the closest neighbouring clusters), and does not generalize to other clustering criteria. Furthermore, it does not account for any interaction between this particular measure and any additional aspects of problem difficulty, such as cluster eccentricity.

In [13], datasets of a defined difficulty are evolved for supervised classification problems. Apart from the emphasis on supervised learning (where, arguably, measures of problem difficulty are more easily derived), a key difference to our work is the method of data generation. Instead of using a generative model, Macià et al. use a sampling approach where an individual corresponds to a subset of a pre-existing dataset, where these subsets are then evolved towards a desired level of problem difficulty. While this approach elegantly avoids the bias introduced through the use of a specific generative model, it may be limited in its ability to generate data sets covering previously unseen portions of the instance space.

In the following, we describe an evolutionary cluster generator designed to overcome these limitations. Our generation of valid covariance matrices closely follows the approach described in [16] and the corresponding *R* package.[1] However, different from previous work we embed this mechanism into an EA to directly optimize this (and all other aspects) of our generating model. Furthermore, we propose using an established internal validity index as a dimensionless measure of problem difficulty. The adoption of an evolutionary framework allows for easy modifications and future extensions, particularly with design choices such as different (or multiple) measures of problem difficulty or the generating mechanism.

A practical guide to using the tool can be found with the (Python) code online[2]; in this paper, we will discuss how the generator works, compare it to existing generators, and show the effect of trading off different aspects of problem difficulty.

3 EVOLVING CLUSTERS

We treat the task of evolving clusters as a constrained optimization problem, where the constraints can be controlled to introduce additional elements of cluster difficulty into the generated datasets. Essentially, the objective function and all constraints can each be interpreted as a specific measure of problem difficulty, with an understanding that the complexity of a particular dataset is, ultimately, algorithm-specific and established by the performance of different clustering algorithms. An EA is used to tackle this problem, the details of which will follow.

3.1 Representation

Each individual in the population represents a single dataset of dimensionality D, and (at present) each cluster is described by a multivariate Gaussian. For simplicity of illustration, we view each Gaussian as being defined by two (sets of) decision variables: a $D \times 1$ vector describing its mean, μ, and the associated $D \times D$ covariance matrix, Σ. As the number of clusters, K, is a parameter to the tool, the length of the genotype is thus $2K$ (sets). Evidently, the size of the individual sets varies with the dimensionality of the data. A 2D example of this representation and the clusters that it encodes can be seen in Figure 2. Here we can see both one eccentric and one more spherical cluster with their corresponding covariances.

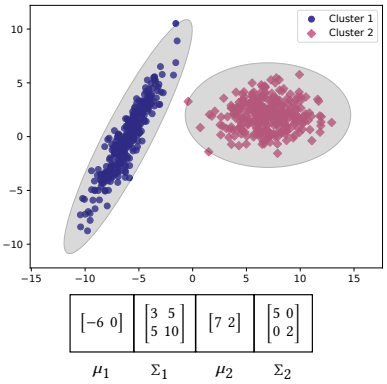


Figure 2: Example representation of a two-cluster problem.

3.2 Objective

The choice of objective function is one of the most delicate aspects of designing a generator for synthetic clustering benchmarks. While

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

[2] Available at: https://github.com/sea-shunned/hawks

an optimization process is required to address the limitations of trial-and-error placement of cluster structures, and to nudge cluster parameters towards challenging features, it is clear that the choice of any particular cluster criterion will bias the dataset towards algorithms based on similar criteria. Additionally, any dataset derived by optimizing a single cluster quality index will become trivial to partition for many algorithms, as top scores for such measures are easily obtained for data sets with isolated spherical clusters.

Our approach to addressing this dilemma is three-fold:

- To use an internal validity index that provides a dimensionless and well-defined measure of problem difficulty, so that a target difficulty value can be defined to optimize.
- To directly consider the trade-off between multiple criteria of problem difficulty using the principle of stochastic ranking.
- To adopt an optimization approach that is flexible with respect to the optimization criterion considered (which can later allow for the generation of a diverse benchmark with known, but diverse, biases).

The cluster validity index adopted in the present paper is the silhouette width [19], which obtains values in a well-defined range. Unlike some other indices, it attempts to provide a straightforward "absolute" interpretation of the amount of structure detected that is roughly comparable across data sets of similar dimensionality. Conceptually, the silhouette width considers aspects of intra-cluster compactness and inter-cluster separability in its calculation. Concrete details of its calculation are as follows:

The silhouette width for a single data point i is defined as:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{1}$$

where

$$a(i) = \frac{\sum_{j \in C_k} \sigma(i,j)}{|C_k|}; \qquad i \neq j;\ i \in C_k \tag{2}$$

$$b(i) = \min_{\forall k \in C} \frac{\sum_{j \in C_k} \sigma(i,j)}{|C_k|}; i \notin C_k \tag{3}$$

and where C is the set of all clusters, C_k is the kth cluster with $|C_k|$ as its cardinality, and $\sigma(i,j)$ is the distance between points i and j. Here, $a(i)$ represents the average intra-cluster distance for data point i and is the average distance from i to all other points in the cluster. The second term, $b(i)$, represents the inter-cluster distance; for data point i this is defined as the minimum of the average distances to all points in every other cluster.

The average silhouette width can then be calculated for a whole individual (i.e. dataset):

$$s_{all} = \frac{1}{N} \sum_{i=1}^{N} s(i) \tag{4}$$

where N is the number of data points. This provides a value in the range $[-1, 1]$, where the maximum represents very compact and well-separated clusters. A negative silhouette width value indicates that points in different clusters are not well-separated (and their membership should be changed).

One issue with the silhouette width is its complexity $O(DN^2)$, where D is the number of dimensions, which has made its use as an optimization objective problematic [26]. As the points in a dataset only change when one of the genetic operators perturbs the mean or covariance of a cluster, this facilitates partial evaluation of the silhouette width. As such, this complexity is worst-case, which occurs only when every cluster in the individual has been changed.

To better obtain solutions of a desired silhouette width, we specify a target silhouette width (s_t), and our objective (fitness) is to minimize the absolute difference between s_t and s_{all}:

$$\min\ f(\mu_1, \Sigma_1, \ldots, \mu_K, \Sigma_K) = |s_t - s_{all}| \tag{5}$$

where K is the total number of clusters.

3.3 Initialization

Each individual is initialized with a random mean and covariance. The mean is sampled uniformly over the half-open interval $[0, 1)$. The variances of the covariance matrix are sampled uniformly over the interval $\left[0, \frac{1}{2}\right)$ to encourage overlap (discussed below). To create the covariances, a random orthogonal matrix is drawn from the Haar distribution [24], i.e. a random rotation matrix is created and transforms the cluster to a random orientation.

Although our generator provides the ability to define the number of data points in each cluster (both precisely and by density), in this paper we randomly assign the size of each cluster such that they sum to a pre-defined size N. The sizes are then fixed for the remainder of the run. This parameter could be added as an additional decision variable in future work.

Initializing with separated clusters can result in the optimizer exploiting particularities of the objective function, e.g. for the silhouette width a moderate reduction can be achieved by adjusting the positioning of individual pairs alone, without consideration of the overall structure of the data set.[3] By initializing with overlapping clusters, we reduce the likelihood of encountering local optima corresponding to such peculiarities.

3.4 Variation operators

3.4.1 Crossover. A standard uniform crossover scheme is utilized, and is illustrated in Figure 3. Each μ and Σ decision variable (set) is swapped independently. Treating both variables as a single unit (i.e. only allowing them to be swapped together) can result in a lack of diversity, particularly when small K values are used. The probability that two individuals undergo crossover is 0.7.

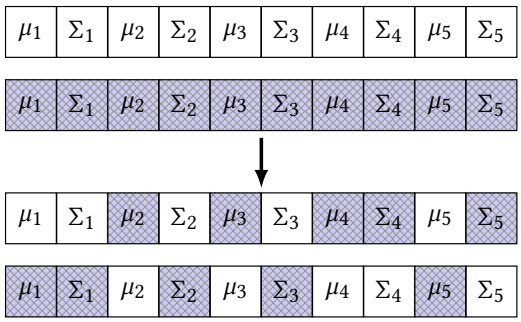


Figure 3: Uniform crossover of individual decision variables

[3] As an example, a moderate silhouette width can be achieved by simultaneously having a pair of clusters on top of each other while remaining well-separated from other pairs.

3.4.2 Mutation. In designing a mutation operator, we needed to ensure that the perturbation was both geometrically meaningful and provided a significant random adjustment to the cluster structure. To provide a meaningful mutation, the mean and covariance parts of the genotype are handled separately. Mutation rates for both types of decision variables are set separately, so that on average one mean and one covariance mutation occurs per individual mutation (i.e. the mean and covariance have a $\frac{1}{K}$ chance of mutation).

To mutate the mean, the new mean (μ_k^{new}) is sampled from a Gaussian around the current mean, i.e. $\mu_k^{new} \sim \mathcal{N}(\mu_k^{curr}, s)$, where s is the width (variance) of the Gaussian. This effectively shifts the cluster in a random direction.

As the decision variable μ_k is described by a D-dimensional vector, it effectively corresponds to D individual decision variables. To avoid over-inflated mutation rates, the probability of shifting the mean in a particular dimension is $\frac{1}{D}$. As the dimensionality of a data set increases, there are, on average, more directions (mutations) that move a cluster centroid away from the other clusters than towards them. In future work, we may consider explicit ways of mitigating this effect, which can result in unintended drift towards high values of the validity index.

Mutating the covariance involves the creation of a transformation matrix, comprised of a scaling and rotation matrix. Performing an eigendecompostion on the covariance matrix, we can view the scaling matrix as randomly modifying the eigenvalues (eccentricity) and the rotation matrix as modifying the eigenvectors (orientation).

For the creation of the scaling matrix, the use of an additive constant or randomly sampled multiplicative scalar leads to a biased distortion over time (creating only highly eccentric clusters). To avoid this, the scaling matrix modifies the variances but maintains the overall determinant of the covariance matrix, preserving the volume of the (hyper)ellipsoid. The rotation matrix is created in the same way as the initial cluster orientation, but raised to a fractional power to ensure the rotation is a perturbation, maintaining similarity to the previous orientation.

3.5 Constraints

Two constraints are currently included to introduce additional aspects of problem difficulty, providing additional control of the complexity of cluster structures and the direct modulation of potential trade-offs. While our objective function (using an internal validity index) is designed to provide direct and quantifiable control regarding the overall (global) difficulty of a dataset, the proposed constraints help consider additional local aspects of cluster structure that can drive difficulty. In view of the representation and generating model currently adopted in our tool, we have identified cluster overlap and cluster shape as primary candidates for these constraint measures. Future work may consider additional aspects such as the balance of cluster sizes or other cluster shapes.

3.5.1 Eccentricity constraint. As the rotation matrix and (diagonal) covariance matrix are kept separate, the variances/eigenvalues of the covariance matrix directly specify the eccentricity of the cluster. To control the extent of this eccentricity, we calculate the eigenvalue (λ) ratio between the smallest and largest variance. For a given individual, this is calculated as:

$$\lambda^{ratio} = \max_{\forall k \in \{1,\ldots,K\}} \frac{\max_{\forall i \in \{1,\ldots,D\}} \Sigma_{ii}^k}{\min_{\forall i \in \{1,\ldots,D\}} \Sigma_{ii}^k}. \tag{6}$$

Controlling the amount of eccentricity that leads to an infeasible solution can allow for the creation of elongated clusters, which can introduce challenges for both the silhouette width and clustering algorithms that optimise compactness (e.g. K-Means).

3.5.2 Overlap constraint. Noisy real-world data can have clusters that overlap, which affects clustering performance. Robustness to overlap, however, varies significantly between algorithms. By constraining (or enforcing) an amount of overlap in the datasets a wider diversity of cluster structures can be created with an additional layer of complexity to the cluster structure defined by the s_t.

The calculation of the overlap itself can also result in a bias. For example, in [4] the authors determine that a point overlaps if its own centroid is further away than a point in another cluster. Such a constraint may potentially prevent the generation of highly eccentric clusters, even where these are spatially well-separated.

We use a nearest neighbour method to calculate overlap, defined as the percentage of data points whose nearest neighbour is a member of a different cluster. This value is in the range [0, 1], though for two Gaussians with identical μ, Σ, and sampling density, the expected overlap is 0.5. Values higher than this can be achieved when there are multiple overlapping clusters, and clusters of different densities. The overlap is defined formally as:

$$overlap = 1 - \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{C^i}(i_{nn}) \tag{7}$$

where C^i is the cluster that data point i belongs to, i_{nn} is i's nearest neighbour, and $\mathbb{1}$ is the indicator function defined as:

$$\mathbb{1}_{C^i}(i_{nn}) := \begin{cases} 1, & i_{nn} \in C^i \\ 0, & i_{nn} \notin C^i \end{cases}$$

As these constraints are conflicting and can be impossible to completely satisfy, many infeasible (but useful) solutions are created during a run of our EA. Stochastic ranking [20] is used to trade-off selecting for fitness and level of feasibility. This is achieved using the parameter P_f, which is the probability that the objective function rather than the constraint penalties will be used for comparisons between infeasible individuals. This parameter influences whether the search favours feasibility or optimality. In accordance with [20], the actual penalties for the individuals are calculated using the quadratic loss function.

3.6 Evolutionary algorithm

A low population size (of 10) is used, and the EA is run for 100 generations. Although this has not been tuned, preliminary experiments showed these values to be sufficient. Figure 4 illustrates how a typical run of the generator (using 60 generations) looks in terms of the actual dataset that is produced. A simple, well-separated dataset is generated using $s_t = 0.9$. As previously mentioned, the clusters are initialized with significant overlap – even after 10 generations the clusters still look almost completely on top of each

other. The fitness drops over time, slowly separating the clusters as shown at generation 30, and then even more after 50 generations.

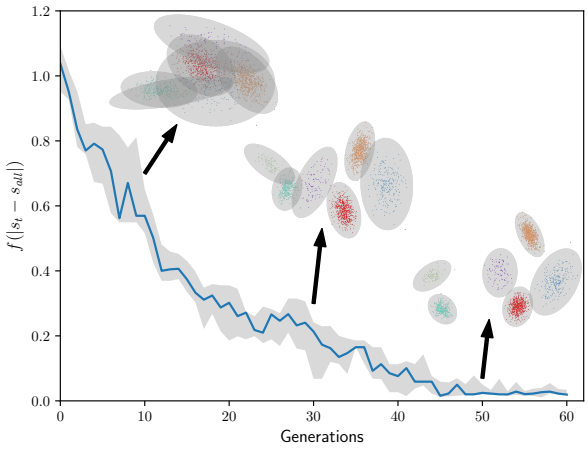


Figure 4: The line shows the median fitness (where $s_t = 0.9$) of the population. The data points and respective Gaussian (with width 3σ) of the best individuals at generations 10, 30, and 50 are also shown.

A standard binary tournament is used to select the parents for crossover and mutation. Stochastic ranking is used to sort the combined pool of the current population and the offspring, from which the best 20% of individuals are selected for the next generation. The remainder are selected randomly from the combined pool.

4 EXPERIMENTS & RESULTS

In this section we illustrate how our generator can be used to generate datasets trading off aspects of problem difficulty and how these adjustments impact the performance of standard clustering techniques. To do this, we first report an experiment designed to identify relevant values of target difficulty for different dimensionalities. We then show how controlled variation of the stochastic ranking parameter allows us to generate datasets associated with different trade-offs between the silhouette width and cluster overlap to produce datasets with pronounced differences in difficulty.

We further provide a comparison of the datasets generated in these experiments to those produced by existing generators. We do this both by considering the location of individual benchmarks within a simple instance space, and by contrasting diversity in performance of clustering algorithms across different benchmark sets. Unless stated otherwise, all experiments use 20 independent runs of the generator, and the best (i.e. closest to the target s_t) individual from each run is selected for analysis.

To consider clustering performance on the benchmark data, the following algorithms are used: K-Means++ [2], single- and average-linkage, and Gaussian mixture models (GMM), as available in *scikit-learn* [15] with default parameters. Each algorithm is given the true number of clusters. As hierarchical clustering algorithms (particularly single-linkage) are prone to identifying singleton clusters and sometimes fare better with a higher number of clusters, these algorithms were also run with an inflated number of clusters ($2K$).

To measure performance, we use the Adjusted Rand Index (ARI) [10], an external cluster validity index that uses the true cluster membership. The ARI is bound in the range $[0, 1]$, where 1 represents a perfect assignment of labels to the data points, and 0 equates to random assignment.

4.1 Defining target difficulty

Compared to alternative validity indices, the silhouette width has a straightforward interpretation as it abstracts from the scale of a particular data set. Nevertheless, as mentioned previously, the measure is not entirely insensitive to changes in the dimensionality of a data set due to the curse of dimensionality. At high dimensions, distances between points become increasingly similar, deflating the values typically associated with well-separated cluster structures.

Our first experiment was therefore concerned with identifying a silhouette width range of interest for a given dimensionality. Specifically, for desired dimensionality D, we aim to identify which values of the silhouette width are associated with non-trivial datasets, i.e. datasets that are sufficiently complex for a trade-off between different aspects of cluster quality to exist. Here, we use the silhouette width and our overlap constraint in making this assessment.

To illustrate this, we generate two clusters at both a low (2D) and high (20D) dimensionality. The two clusters start with the same mean, and are perfectly hyperspherical (i.e. $\Sigma = \mathcal{I}_D$), which with their equal density should result in *overlap* ≈ 0.5. The clusters are gradually separated by moving the cluster means in an equal and opposite direction along a single axis.

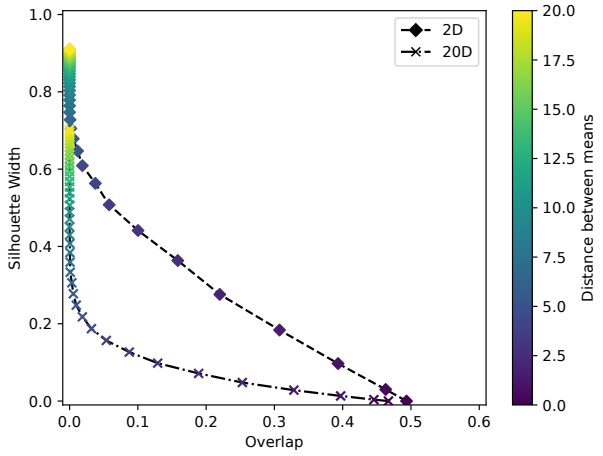
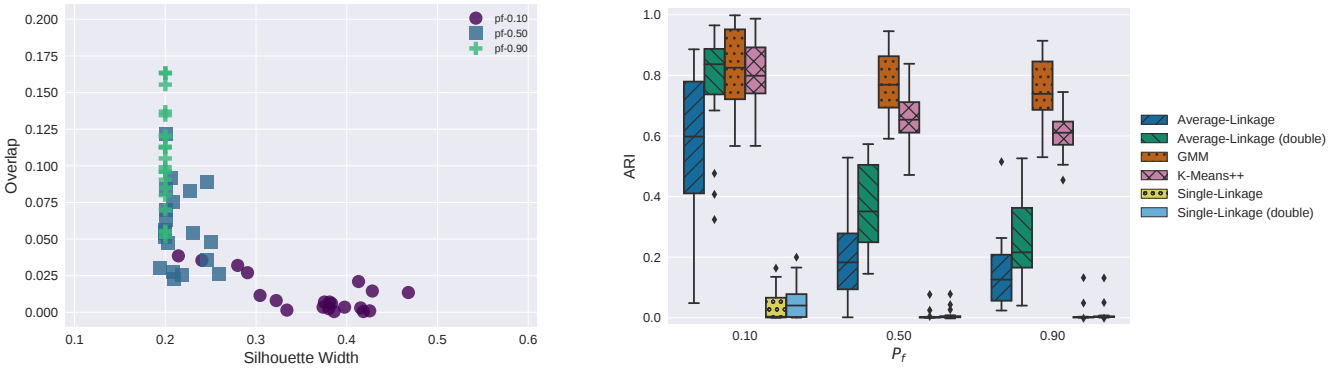


Figure 5: Interaction between s_{all} and overlap for two drifting Gaussians at low (2D) and high (20D) dimensions.

As may be expected, Figure 5 shows that both the silhouette width and the overlap decrease as the clusters move apart. However, in 20D, the overlap decreases rapidly for a small increase in the silhouette width. Conversely, in 2D, the rate of change for the silhouette width and overlap is more similar. It is clear that, in higher dimensions, even well separated clusters (with no actual overlap) may not be associated with a silhouette width of 0.35 or higher. This behaviour is important to consider when selecting appropriate (and challenging) values for these parameters at the desired dimensionality.



(a) (b)

Figure 6: The scatter plot in (a) shows how the trade-off between the silhouette width and overlap can be managed by the P_f parameter. In (b), we see how this is reflected in a difference in performance of clustering algorithms, where the hierarchical clustering methods favour a bias towards minimizing overlap.

4.2 Exploring the instance space

We use the stochastic ranking parameter P_f to explicitly explore specific areas of interest in the instance space defined by the silhouette width and our measure of overlap. To do so, our overlap constraint is set to an upper bound of 0 (i.e. $overlap \leq 0$), penalizing any overlap. Based on the results from the previous section, we chose our silhouette width target to be $s_t = 0.2$ in 20D and $s_t = 0.6$ in 2D. We then select P_f values of $\{0.1, 0.5, 0.9\}$, representing a favouring of the constraints (pushing towards no overlap), no bias, and favouring the objective respectively.

In Figure 6a, the actual silhouette widths and overlap obtained in 20D are shown for the three P_f values. There is a clear trade-off between the silhouette width and the overlap, the search of which is controlled by the P_f value. In such a conflicting scenario, the overlap constraint acts as a second objective, weighted by P_f, and variation of P_f allows the generation of datasets in different regions of the instance space.

Figure 6b shows the associated variation of clustering performance for these setups. There are pronounced differences in the absolute and relative performance of traditional clustering algorithms. Most notably, average-linkage emerges as a serious contestant for the benchmarks with limited overlap ($P_f = 0.1$), but its performance declines severely as the extent of the overlap increases (progressively from $P_f = 0.5$ to $P_f = 0.9$).

4.3 Comparing generators

To get an idea of the diversity of performance of algorithms on our benchmarks, we generated a small batch of datasets using the parameters highlighted in Table 1. While this represents a narrow range of the capabilities for our generator, it is realistic to desire minimal overlap and not consider eccentricity.

Taking the best from each of 20 runs, across 12 unique combinations of parameters, 240 datasets were generated in total. To isolate the effect of optimizing the silhouette width by itself, the λ^{ratio} is unconstrained and overlap is discouraged (i.e. constrained as $overlap \leq 0$), and the search is unbiased between the objectives and constraints ($P_f = 0.5$).

Table 1: Generator Parameters

Parameter	Symbol	Value(s)
# datapoints	N	2000
# clusters	K	$\{5, 30\}$
# dimensions	D	$\{2, 20\}$
Stochastic ranking parameter	P_f	0.5
Fitness	$f()$	$\{0.2, 0.5, 0.8\}$
Eigenvalue ratio	λ^{ratio}	≥ 1
Overlap	$overlap$	≤ 0

These datasets were compared against the clustering benchmarks of HK used in [6] (160 datasets), and QJ used in [16] (243 datasets). HK's datasets consist of 80 from the 'gaussian' generator with $D \in \{2, 10\}$ and $K \in \{4, 10, 20, 40\}$, and 80 datasets from the 'ellipsoid' generator with $D \in \{50, 100\}$ and $K \in \{4, 10, 20, 40\}$, covering a range of settings.

The datasets produced by Qui & Joe define three levels of separation: 'close structure', 'separated', and 'well-separated'. A range of dimensionality is used, from 5 to 24 dimensions, and $K \in \{3, 6, 9\}$. They specify differing levels of noisy variables to add further complexity. Further details of these datasets can be found in their respective papers. The results of running the aforementioned clustering algorithms on these datasets can be seen in Figure 7.

A wide spectrum of performance is obtained by all generators, showing that (at least for some algorithms) a diverse set of cluster structures is generated. Note that aggregating across this many diverse datasets is unhelpful for empirically comparing algorithmic performance, but it is useful for evaluating the diversity (of the clustering challenge) that the generator can create.

A broad spectrum of performance is obtained by HK's generator for all algorithms aside from single-linkage. The noticeably lower medians for our generator indicate a higher degree of difficulty, and

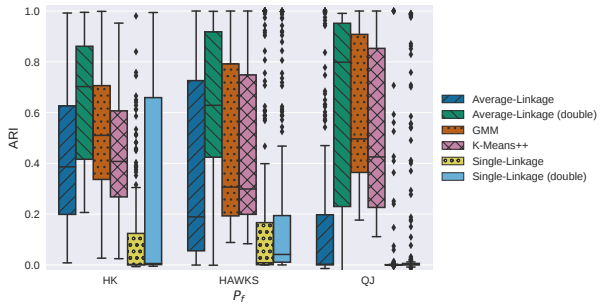


Figure 7: Comparison of the HK [6], HAWKS, and QJ [16] generators. A greater variance in performances indicates a wider diversity of datasets, and a lower median indicates a higher difficulty across the datasets for that algorithm.

larger variance indicate a more diverse spread of datasets. Despite this advantage, the benchmarks considered in this experiment do not include datasets favouring the single link algorithm. We believe that this is due to the limitation of using a single setting ($P_f = 0.5$) for the stochastic ranking parameter only, and can be overcome by including a lower choice in our experiments. As evident from Figure 6, this is required to push for clusters with minimal overlap. Our datasets do contain clusters with highly eccentric shapes ($\lambda^{ratio} \geq 1000$), and this, in combination with little overlap, should produce benchmarks that favour the single link algorithm.

For QJ's generator, performance for average- and single-linkage is even worse, and there is a bias in performance towards K-Means++ and GMM. This indicates that there is a lack of eccentricity in the clusters, favouring these compactness-based clustering methods. It should be noted here though that their generator can produce more eccentric clusters, but in the suite specified in [16] eccentricity was limited to an equivalent maximum of $\lambda^{ratio} = 10$. It should further be noted that even their most challenging datasets ('close structure') did not degrade performance as significantly as datasets obtained from the optimization to a low silhouette width.

To illustrate the generators in a similar way to the 'instance space' work of Smith-Miles [22], Figure 8 shows a plot of the generators according to the silhouette width, *overlap* measure, and dimensionality. To visualize the data, PCA has been applied to reduce the space to 2D. Both our generator and QJ's cover a range of the space, whereas HK's generator favours low overlap and higher silhouette widths, with the separation across the second principal component largely due to the higher dimensionality of their data. Although this a simple instance space, there is a clear difference in coverage between the generators, which is encouraging for our approach.

5 CONCLUSION AND FUTURE WORK

Synthetic data generators play an important role in creating benchmark datasets that can help assess the performance of new algorithms, and meaningfully compare different techniques. A solid understanding of the properties and complexity of synthetic data facilitates the design of robust evaluations of algorithms, providing insights into their specific strengths and weaknesses.

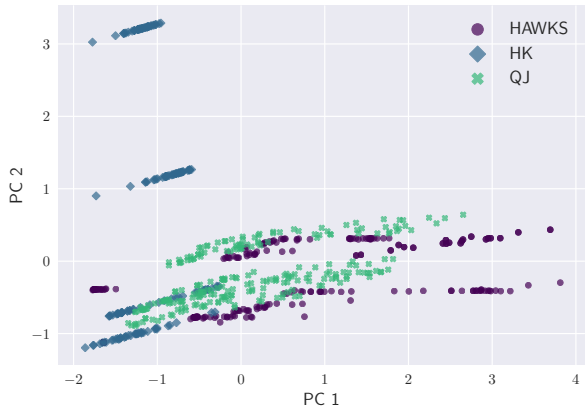


Figure 8: Comparison of the generators in a simple instance space. Greater coverage of the space indicates a wider diversity of datasets. PCA is used to reduce the silhouette width, *overlap*, and number of dimensions in the datasets to 2D.

We have created HAWKS to try and generate datasets of a defined level of difficulty for the problem of cluster analysis. Our experiments demonstrate that, even for a simple setup (involving three aspects of problem difficulty), we are able to generate datasets that exceed the diversity of existing benchmarks, triggering significant changes in the performance of standard clustering methods. In future work, by using state-of-the-art clustering algorithms we can better assess the nuanced difficulty of our generated datasets. A core advantage of our generator is the use of an evolutionary computation framework, which allows for the easy future replacement of individual components of our tool. The sole use of Gaussian clusters is an obvious limitation of the current prototype; the creation of non-convex clusters in a similarly parameterized way would further facilitate empirical comparisons between algorithms.

Our current method uses the mechanism of stochastic ranking to illustrate the trade-off between different aspects of problem difficulty. The consideration of multi-objective optimization techniques, to support this exploration, seems like an obvious extension. Furthermore, our experiments are currently limited to the use of a single validity index. As discussed in earlier sections of the paper, a fair evaluation of clustering algorithms will require the definition of an instance space (and benchmark set) obtained through the consideration of a much larger number of complementary validation techniques. Future work will also need to look at the scalability of this generator, particularly when extending to multiple objectives.

ACKNOWLEDGMENTS

Cameron Shand acknowledges PhD funding support from the EPSRC Manchester Centre for Doctoral Training in Computer Science (EP/I028099/1).

REFERENCES

[1] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M. Pérez, and Iñigo Perona. 2013. An extensive comparative study of cluster validity indices. *Pattern Recognition* 46, 1 (2013), 243 – 256. https://doi.org/10.1016/j.patcog.2012.07.021

[2] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1027–1035.

[3] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[4] Pasi Fränti and Sami Sieranoja. 2018. K-means properties on six clustering benchmark datasets. *Applied Intelligence* 48, 12 (2018), 4743–4759.

[5] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 416–419.

[6] J. Handl and J. Knowles. 2005. Improvements to the scalability of multiobjective clustering. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 3. 2372–2379 Vol. 3. https://doi.org/10.1109/CEC.2005.1554990

[7] Julia Handl and Joshua Knowles. 2006. *Multi-Objective Clustering and Cluster Validation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 21–47. https://doi.org/10.1007/3-540-33019-4_2

[8] Daniel S Himmelstein, Casey S Greene, and Jason H Moore. 2011. Evolving hard problems: generating human genetics datasets with a complex etiology. *BioData mining* 4, 1 (2011), 21.

[9] John N Hooker. 1995. Testing heuristics: We have it all wrong. *Journal of heuristics* 1, 1 (1995), 33–42.

[10] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218.

[11] Yang Lou, Shiu Yin Yuen, and Guanrong Chen. 2018. Evolving benchmark functions using kruskal-wallis test. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 1337–1341.

[12] Núria Macià and Ester Bernadó-Mansilla. 2014. Towards UCI+: a mindful repository design. *Information Sciences* 261 (2014), 237–262.

[13] Núria Macià, Albert Orriols-Puig, and Ester Bernadó-Mansilla. 2010. In search of targeted-complexity problems. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 1055–1062.

[14] Mario A Munoz, Laura Villanova, Davaatseren Baatar, and Kate Smith-Miles. 2018. Instance spaces for machine learning classification. *Machine Learning* 107, 1 (2018), 109–147.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[16] Weiliang Qiu and Harry Joe. 2006. Generation of random clusters with specified degree of separation. *Journal of Classification* 23, 2 (2006), 315–334.

[17] Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M Quiroz. 2011. Internal versus external cluster validation indexes. *International Journal of computers and communications* 5, 1 (2011), 27–34.

[18] John R Rice. 1976. The algorithm selection problem. In *Advances in computers*. Vol. 15. Elsevier, 65–118.

[19] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

[20] T. P. Runarsson and Xin Yao. 2000. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation* 4, 3 (Sep. 2000), 284–294. https://doi.org/10.1109/4235.873238

[21] Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. 2014. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research* 45 (2014), 12–24.

[22] Kate Smith-Miles and Simon Bowly. 2015. Generating new test instances by evolving in instance space. *Computers & Operations Research* 63 (2015), 102–113.

[23] Kate Smith-Miles and Leo Lopes. 2012. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research* 39, 5 (2012), 875–889.

[24] G. W. Stewart. 1980. The Efficient Generation of Random Orthogonal Matrices with an Application to Condition Estimators. *SIAM J. Numer. Anal.* 17, 3 (1980), 403–409. http://www.jstor.org/stable/2156882

[25] Jano I van Hemert. 2006. Evolving combinatorial problem instances that are difficult to solve. *Evolutionary Computation* 14, 4 (2006), 433–462.

[26] Lucas Vendramin, Ricardo JGB Campello, and Eduardo R Hruschka. 2010. Relative clustering validity criteria: A comparative overview. *Statistical analysis and data mining: the ASA data science journal* 3, 4 (2010), 209–235.

[27] David H Wolpert. 1996. The lack of a priori distinctions between learning algorithms. *Neural computation* 8, 7 (1996), 1341–1390.

[28] David H Wolpert and William G Macready. 1997. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* 1, 1 (1997), 67–82.