Proceedings of the ASME 2021
International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
IDETC/CIE 2021
August 17-20, 2021, Online, Virtual

DETC2021-69328

AUTOMATICALLY DISCOVERING MECHANICAL FUNCTIONS FROM PHYSICAL BEHAVIORS VIA CLUSTERING

Kevin N. Chiu
Dept. of Mechanical Engineering
University of Maryland
College Park, Maryland
Email: knchiu@terpmail.umd.edu

David Anderson
Engora, Inc.
Boston, Massachusetts

Mark D. Fuge*
Dept. of Mechanical Engineering
University of Maryland
College Park, Maryland
Email: fuge@umd.edu

ABSTRACT

Computational design methods provide opportunities to discover novel and diverse designs that traditional optimization approaches cannot find or that use physical phenomena in ways that engineers have overlooked. However, existing methods require supervised objectives to search or optimize for explicit behaviors or functions — e.g., optimizing aerodynamic lift. In contrast, this paper unpacks what it means to discover interesting behaviors or functions we do not know about a priori using data from experiments or simulation in a fully unsupervised way. Doing so enables computers to invent or re-invent new or existing mechanical functions given only measurements of physical fields (e.g., pressure or electromagnetic fields) without directly specifying a set of objectives to optimize.

This paper explores this approach via two related parts. First, we study clustering algorithms that can detect novel device families from simulation data. Specifically, we contribute a modification to the Hierarchical Density-Based Spatial Clustering of Applications with Noise algorithm via the use of the silhouette score to reduce excessively granular clusters. Second, we study multiple ways by which we preprocess simulation data to increase its discriminatory power in the context of clustering device behavior. This leads to an insight regarding the important role that a design's representation has in compactly encoding its behavior.

We test our contributions via the task of discovering designs that function as fluidic logic gates. We generate synthetic data that mimics fluidic devices and show that our proposed contributions better discover logic gates, as measured by adjusted Rand score. Specifically, combining our Resolution Selection preprocessing and principal component analysis resulted in the highest and tightest spread of adjusted Rand scores on our tested datasets. This opens up new avenues of research wherein computers can automatically explore different types of physics and then derive new device functions, behaviors, and structures without the need for human labels or guidance.

1 Introduction

One output of engineering design is to manipulate geometry and material in space-time (often called a device's *structure*) to modify something about nature's current state via physical fields (*i.e.*, a device's *behavior*) in service of benefiting humanity in a specific way (*i.e.*, a device's *function*). For example, devices called logic gates all perform the same useful *function — e.g.*, performing a Boolean operations (such as AND, OR, XOR) between one or more inputs. Despite this seemingly commonplace function, however, logic gates can vary wildly in their physical *behavior*. For example, while familiar AND gates (*e.g.*, using MOSFETs) manipulate electrostatic fields to alter a material's conductivity, there are a wide range of other physical behaviors that can produce this same function: (1) *fluidic* logic gates can use dye flowing into different channels [1],

*Address all correspondence to this author.

pressure-activated gates [2], or even bubble-based gates [3] to create fluidic circuits used in sensor or lab-on-a-chip applications; (2) *photonic* gates can guide waves through crystals [4]; (3) *chemical* diffusion can drive AND behavior in synthetic biology circuits [5]; and (4) *trapped ions* [6] or *laser* pulses [7] can combine to form Toffoli gates, useful for higher qubit quantum circuits.

In principle, all of these devices can have the same function — performing Boolean operations — though they implement that function via drastically different physical behaviors. This paper addresses how one might discover a variety of similarly functioning devices automatically. Specifically, we seek algorithms that can identify "interesting" physical behaviors that emerge from a set of physics and a given design space, but without requiring human guidance, intervention, or explicit direction regarding what behaviors or functions to look for. For example, put more succinctly, without telling an algorithm that we want something that acts like a logic gate, can it nevertheless automatically discover the existence of such devices by only directly observing the physical behavior of various designs? Can it derive novel families of devices that perform functions that the algorithm does not know about *a priori*? Answering such questions would enable the automated discovery of engineered components for new physics regimes where humans have yet to explore rigorously or even via trial-and-error — *e.g.*, in spacecraft components in extreme environments where existing devices break down such as in landers on Venus [8] or hypersonic flight regimes, or in adapting engineering design methods to novel domains like synthetic biology or quantum computing.

This paper addresses such concerns by studying two interrelated questions: (1) how do we discover groups of designs whose behavior is "similar enough" to constitute a family of related functions, and (2) how do we compactly represent a design's behavior? This paper's key idea is that by combining large-scale, automated simulation of different physics across (appropriately randomized) design spaces with new ways of preprocessing and clustering the resulting simulation data, we can automatically identify unique functional families of devices. Specifically, the key contributions of this paper are:

1. We modify the Hierarchical Density-Based Spatial Clustering of Applications with Noise ("HDBSCAN") algorithm's persistence measure and use the silhouette score [9] to select hyperparameters. When combined with the below contribution, this method, called the Silhouette-modified HDBSCAN ("SHDBSCAN"), produces clusterings with generally higher adjusted Rand scores than tested baseline methods on behavior clustering tasks.
2. We propose two heuristics that produce behavior vectors that provide better clustering of fluidic components than unpreprocessed data when applied to synthetic examples that mimic boundary-sampled simulation solution fields [10]. We identify a subset of preprocessing steps that result in the highest adjusted Rand scores on our tested data.

To demonstrate the value of these contributions, we show that we group synthetic fluidic devices with similar logic gate behavior together without explicitly stating objectives or optimizing for logic gate behavior. Stated more plainly, interesting logic-gate behavior "falls out naturally" from our above behavior representation and clustering, even though we do not explicitly instruct the algorithm to find those specific behaviors.

2 Related Work

This paper contributes to three main areas: describing behavior of components from simulation data, clustering of behavior to discover groups of components with similar behavior, and furthering our methods of functional discovery. Although we use fluidic logic gates components as example cases, our contributions are generalizable to other types of simulation data as well.

2.1 Discretizing Continuous Data

To use the plethora of machine learning ("ML") methods that have been studied, it is often necessary to vectorize — or put into a vector — the data on which the algorithm will train. Although some applications have very clear ways to vectorize data (*e.g.*, setting features for housing price such as number of floors or square footage), other applications require less obvious set up (*e.g.*, in text documents, using term frequency and inverse dataset frequency [11, 12]).

We run into some challenges exemplified by our sample application of clustering on fluidic logic gates. In the first, fluid simulations generally produce discrete data, though they model continuous fields (*i.e.*, continuous functions). By itself, this is not necessarily an issue; converting from a continuous field to a discrete one is reasonably well-studied in the field of functional data analysis in a process called *quantization*; Wang *et al.* [13] discuss many of the methods used in quantizing data, as well as several methods for clustering such data. However, although these approaches are used in a wide range of applications (DNA in adipose cells [14], time-course gene expression [15], and craniofacial growth of male rats and lung function growth [16], to name a few), these discretization methods also generate high-dimensional representations to capture the behavior of the physical phenomena. This dimensionality poses problems for training ML algorithms on possibly sparse data, invoking the curse of dimensionality [17].

Specifically in this paper, we can quantize continuous physical phenomena faithfully, i.e. capturing the large-scale behavior and fluctuations of the physical fields, but the resulting discretization is often too high-dimensional to extract meaningful device clusterings due to redundant or non-discriminatory

dimensions. Our approach explores combinations of dimension "pruning" — where we heuristically remove redundant and non-discriminatory dimensions — and dimensionality reduction — via Principal Component Analysis ("PCA") and t-distributed Stochastic Neighbor Embedding ("TSNE").

2.2 Clustering

Clustering, a commonly known operation in the field of ML, has been studied extensively in recent years. The goal of clustering algorithms is to assign each data point to the group of data points that are most similar to itself; however, globally optimal clustering is NP-hard [18]. Because of this, many different algorithms have been proposed, some using meta-heuristics like Harmony Search [12, 19], while others are based on swarm intelligence [20, 21], message passing [22], low-dimensional embedding [23], or crawling through regions of high density [24]. Combined with methods as in [25] and above, clustering can be extended to the types of data we would see in fluidic logic gate simulations. However, a straightforward application of these algorithms produces low correlation between clustering and device behavior, specifically from similar devices not being grouped, so we propose a new clustering algorithm — SHDBSCAN (§3.1) — to mitigate these issues without compromising on performance in other clustering applications.

2.3 Functional Discovery

Much of the work in design ideation focuses on human engineers and their processes, though several approaches approach the use of ML in design ideation as well.

By focusing on the human designer, researchers try to understand how human designers work, and then how to improve a human designer's process in designing a product. This begins with how designers represent data internally [26] and use analogies to recall useful previous information [27]. With an internal model in place, designers iterate through solutions until a satisfactory solution is reached. This iterative process may include looking at similar and dissimilar examples to improve feasibility or novelty, respectively [28] or combining previous designs to improve novelty and quantity [29]. However, these methods all presume a human designer; non-human designers — *i.e.*, computers — may not necessarily follow the same internal models or use the same heuristics.

Zhang *et al.* [30, 31] and Camburn *et al.* [32] compare expert and non-expert analyses of design ideas against algorithmically generated analyses, showing that unsupervised ML methods provide analyses comparable to human-generated analyses on text-form descriptions of designs. Gyory *et al.* [33] apply methods from network analytics to describe a semantically designated design space, while Fu *et al.* [34] search for structural forms in patent texts. Similarly, Mikes *et al.* [35] mine repositories for functional decompositions of products in a generalized way. However, these methods focus on textual descriptions of designs, taking advantage of human intuition for the aspects of a design that are important through articulation of these features. Our work differs from this previous work by using numerical descriptions of devices based solely on their measurable physical behaviors, without requiring human intuition or description of what a device does.

Aside from human-centered design methods or intervention by humans, recent work has applied ML to generate new designs or analyze preexisting ones. Generative adversarial networks have been used to increase training datasets for 3-D design problems [36] and to find diverse, high-quality designs in airfoils [37]. Grammars and grammatical structures are used to search for and design tensegrity structures [38], soft robots [39], and cooling channels in a die-casting mold [40]. Additional uses of ML and other heuristics include identifying fasteners (*e.g.*, screws) and non-fasteners [41] and reducing manufacturing costs [42], though these methods are application-specific and may be difficult to generalize. Dering and Tucker [43] use convolutional neural networks to predict *functional quantities*, which, though conceptually similar to the behavior vectors from §3.2, are generated by hand, while our method is fully automated. However, even small differences in representing the same devices can create significant disparities in measurements of design similarities [44]. Our methods differ in that we do not limit ourselves to the representation space of a grammatical language, nor do we presume a repository or previous representation of simulation data. Our proposed methods attempt to minimize redundancy and improve discriminatory power of our behavior vectors.

3 Methods

This paper contributes two key methods, which this section describes in order: (1) SHDBSCAN, a modification to the HDBSCAN algorithm that improves clustering performance; and (2) a series of preprocessing methods to generate behavior vectors from simulation data.

3.1 Silhouette-optimized HDBSCAN

We treat function identification as essentially a clustering problem. That is, we expect that different devices with similar behaviors exist and that grouping those similar behaviors will identify the underlying functions those devices perform. Preliminary results with off-the-shelf clustering algorithms either resulted in clusters of unrelated devices — while related ones were separated into different clusters — or numerous tiny clusters — some including only single devices. Many of these tiny clusters included similar behaviors; as such, we posited that modifying an existing algorithm to force these small clusters to stay together could improve the clustering results. The density-based algorithms, such as HDBSCAN, seemed promising from

our initial results. Although it produced many small clusters, HDBSCAN did not usually cluster together unrelated devices.

The original HDBSCAN algorithm [45] has the following high-level steps:

1. Compute the mutual reachability metric, in effect spreading out regions of low density and compressing regions of high density.
2. Build the minimum spanning tree of the dataset, *e.g.*, using Prim's algorithm [46].
3. Build a hierarchy of how clusters are connected, merging smaller clusters into larger ones as the hierarchy goes up.
4. Condense the cluster tree such that it represents large, persistent clusters that lose points, splitting only when the subsequent clusters are larger than the minimum cluster size.
5. Finally, extract the longest-lived and most persistent clusters by a stability metric, keeping a cluster when its stability is greater than that of its children.

We modify the HDBSCAN algorithm in two ways. First, we change its measure of cluster persistence to a new measure governed by a hyperparameter called the *maximum cluster size*. Secondly, to select this hyperparameter, we select the value that maximizes the silhouette score, which measures the inter- and intra-cluster distances to approximate how well-separated and tightly clustered the resulting clusters are. We call our approach the Silhouette-optimized HDBSCAN, or SHDBSCAN.

First, we consider some value c that we denote the *maximum cluster size*. We start at the root of the cluster tree, representing a clustering where all points are in one cluster. We can thus step down through the cluster tree, clustering our data into smaller and smaller clusters without dropping any data from our clusters. Figure 1 demonstrates how we step down a single-linkage tree for a pedagogically illustrative example.

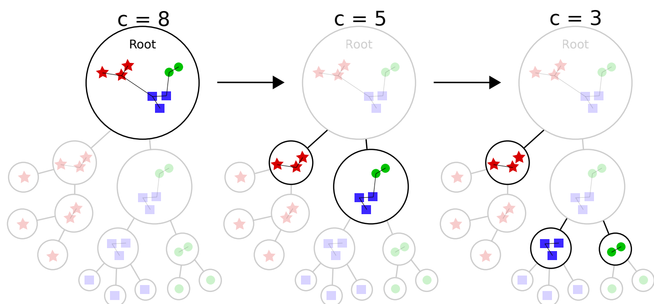


FIGURE 1: At the root node, all nodes are in the same cluster. As we set c smaller, we work our way down through the clusters. By setting $c = 8$, we end our clustering at the root node. If we set $c = 5$, we move down to the next level, where each cluster includes at most 5 nodes. Setting $c = 3$, the left-side cluster is smaller than c, but the right-side cluster must be split once more.

At each node, we calculate the size of that cluster by counting the number of leaves in the subtree rooted at that node; if that cluster size is less than c, we stop at that node and label all leaves of the subtree to be in the same cluster. We then move onto the next node with unlabelled leaves and repeat the process. We continue this process until all points are labeled. This is a different measure of "cluster persistence" than is used in HDBSCAN. However, since our proposed persistence-modified HDBSCAN depends on an appropriate value for c, our algorithm needs to choose this value.

To choose the appropriate max cluster size c, we use the silhouette score. Specifically, we run the persistence-modified HDBSCAN for every integer between $c = 1$ (where every data point is its own cluster) to $c = N$ (where N is the number of points in our dataset, representing all points in a single cluster), calculating the silhouette score for each value of c. We then set c to the value that maximizes the silhouette score.

3.2 Generating Behavior Vectors from Physical Simulations

To identify families of interesting devices, we have to solve three problems:

1. What is behavior, and how do we represent it?
2. How do we minimize redundancy in our behavior vector?
3. How do we remove non-discriminatory values?

As an illustrative example, we discuss a 2-D two-input-one-output fluidic logic gate (Fig. 2).

If we consider that the movement of mass, *e.g.*, fluid velocity, differentiates between fluidic logic gates, then we recognize that the behavior must be related to the fluid velocity. As such, our method of representing the behavior must take into account the fluid velocity. One naive way to represent the behavior of a device is to sample the velocity throughout the domain — *e.g.*, along a grid — and append the velocities into a vector. This provides a large, high-dimensional vector that contains *all* of the information about the behavior. However, this method includes information that may be unnecessary in representing the device's behavior.

To address this inefficiency, we considered what a device's behavior means in terms of its function in a system. We, as engineers, usually care how a device behaves in terms of its relations to its surroundings. In other words, does it matter if a motor spins because of changing magnetic fields or because an elf inside is riding a small bike? Do we need to know what is happening *everywhere*, or can we more compactly — yet comprehensively — represent a device's relationship to its environment by looking only at its interfaces with its environment? Following this line of thought, we sampled points only along the *boundaries* of the device — *i.e.*, where the fluidic logic gate interfaces with the rest of the system. This seemed promising as it

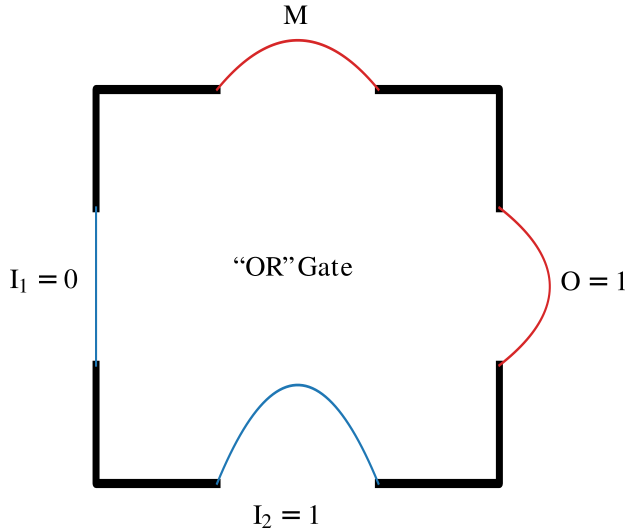


FIGURE 2: Example four-port device that mimics a two-input-one-output logic gate. The blue inputs are $I_1 = 0$ and $I_2 = 1$, representing fluid velocities. The red output is $O = 1$, with the red miscellaneous ("M") gate providing a sink for excess mass in this example. When performing an "OR" operation between I_1 and I_2, we see that O has a positive, if lower, velocity profile. The addition of the red miscellaneous gate at the top provides an outlet for any surplus (or shortage) of fluid to conserve mass. Integrating velocity across all four ports results in 0 net change in fluid flow.

generated a significantly smaller behavior vector. However, we now needed to generate these behavior vectors in a consistent and well-defined manner.

First, we look at the sampling resolution along the border. Sampling more points — *i.e.*, a higher resolution — would give more detailed information about the device, but at the cost of a larger behavior vector. However, we find it is worth looking at more redundant behavior vectors and later pruning out redundant values from our representation, rather than starting with an uninformative behavior vector. Second, we wish to minimize redundancy in the behavior vector we have generated. If a device's behavior vector has slowly changing values, then spatially close points would not provide any additional information about a device, so we can effectively reduce the sampling resolution by removing points from the behavior vector. Thirdly, we want to remove points that, while representative of behavior, may not be useful to discriminate between devices. By removing points with a variance below a given threshold, we can shrink the behavior vector while maintaining its discriminatory value.

This combination of three steps (generating the behavior vector, using a subset of its values, and removing non-discriminatory values) gives us a method by which we can convert the continuous results of a simulation into a vector of discrete values with which we can apply ML methods. The sections below go into detail about the implementation of these steps.

3.2.1 Initial Boundary Resolution To choose boundary points to include in the behavior vector, we consider the edges of the spatial domain. In theory, we can sample any number of points from the simulation data. However, including too many points increases the dimensionality of the clustering problem without increasing our ability to differentiate between clusters. The goal, then, is to include the fewest number of points (to reduce the curse of dimensionality) that still contain enough information to differentiate between different devices.

Choosing the initial sampling resolution is not trivial, though it does correlate strongly with the length scale of phenomena in the device. If the length scale is unknown, then the initial sampling resolution is little more than an arbitrary guess. As a simple heuristic, we suggest a visual inspection of some data points and choosing a resolution twenty times the frequency of the shortest "important" phenomenon, as in Fig. 3. If the length scale is known, we suggest twenty times its minimum frequency as the initial sampling resolution. The specific choice of initial sampling frequency is not critical, so long as it is sufficiently high, since we propose methods to remove redundant and non-discriminatory values in §3.2.2 and §3.2.3.

3.2.2 Resolution Selection Preprocessing We want to capture phenomena without knowing ahead of time the length scale of the phenomena. We estimate the length scale using differences between adjacent points and use this estimation to determine the sampling resolution. This Resolution Selection preprocessing step shortens a current behavior vector by taking every m^*-th point, as defined below.

First, we linearly scale our behavior vectors so each value is between 0 and 1 such that, for our fluidic logic gate example, all of the values related to pressure are scaled together, and all of the values related to velocity are scaled together. If we directly scale the entire behavior vector together, then if, for example, the pressure values were significantly higher than the velocity values, they could dominate the scaling and essentially negate the discriminatory power of velocity values.

Next, for every pair of adjacent points, we calculate the absolute value of their difference, which is guaranteed to be less than 1. We take the largest difference — which approximates the most rapidly changing region — and reciprocate this value, giving a number greater than 1. We call the floor of this value m.

We find the minimum m across all devices and call this m^*. We take every m^*-th point from the original behavior vectors and

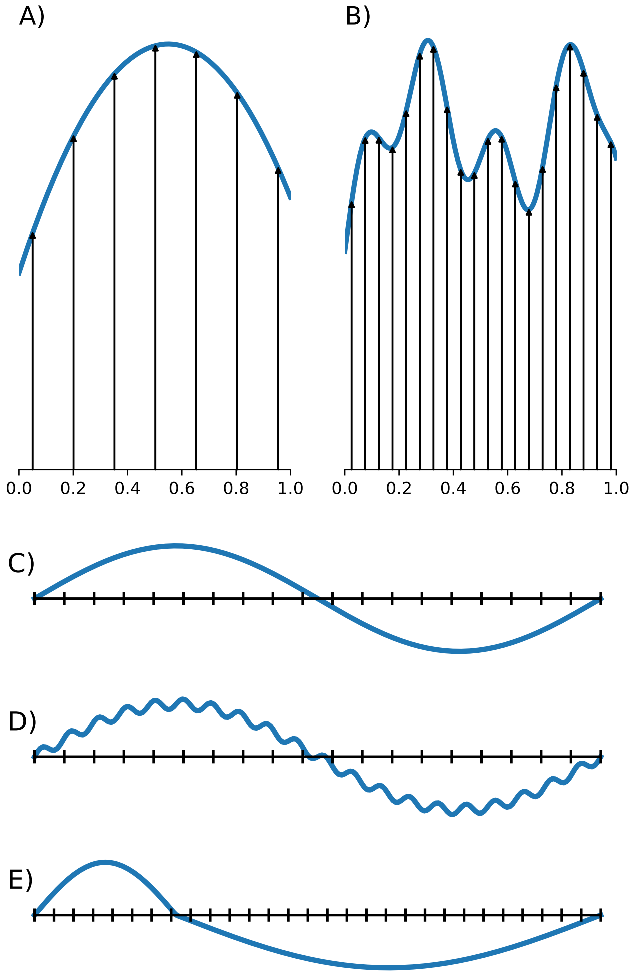


FIGURE 3: A) Phenomena with longer length scales do not require as fine a sampling resolution. B) Phenomena with shorter length scales require finer sampling resolution to capture behavior. C) We suggest a simple heuristic of choosing a resolution that scales with the length scale of "important" phenomena. This is an example of a long phenomenon, D) long phenomenon with shorter noise and E) varying length phenomena, where we use the shortest phenomenon to determine sampling resolution.

use these values as our new behavior vector. This process is exemplified and summarized in Appendix A.

3.2.3 High Variance Filter Preprocessing

Though we can prune the behavior vectors with the Resolution Selection preprocessing in §3.2.2, some values in the behavior vector may still be non-discriminatory, *e.g.*, velocity along a plugged-up port that never has flow going through it. We thus hypothesize that the values in the behavior vectors broadly fall into two categories: low-variance, which are non-discriminatory and need to be removed; and high-variance, which may be discriminatory and preferable to keep.

To remove these low-variance values, we calculate the variance of each value in the behavior vector using Eqn. 1 and remove those that have variances below a certain threshold.

$$V = \frac{1}{N}\sum_{1}^{N}(x_i - \mu)^2 \tag{1}$$

Although we do not formally test choices of this cutoff, we briefly explore how the cutoff value affects the behavior vectors in Appendix B. We find that a cutoff at 0.02 is effective for our behavior vectors of fluidic logic gates, though this value would need to be adjusted for a given dataset.

4 Experiment 1: Testing SHDBSCAN using Synthetic Clusters

This experiment evaluates our SHDBSCAN algorithm by comparing its clustering performance with respect to several classes of baseline clustering algorithms on datasets with known cluster labels.

4.1 Data Generation

For the first experiment, we compare the clustering performance of SHDBSCAN against other well-established clustering methods. We use several well-known datasets of both synthetic and experimental data from the Scikit-learn library and UCI ML repository [47], as discussed in Appendix D.

4.2 Baseline Methods

We explore the Scikit-learn implementations of k-means clustering ("KMeans"), affinity propagation ("AffProp"), spectral clustering ("SpecClustRBF"), and Density-based Spatial Clustering of Applications with Noise ("DBSCAN")[1] [48]. We also test HDBSCAN [49].

We perform 5-fold cross validation with Scikit-learn and hyperparameter optimization using Bayesian Optimization ($\alpha = 1e-4$, $n_{iter} = 5 + 30\times$number of parameters) [50] with the ranges in Table 2 in Appendix C. We do not expect any significant improvements in performance by optimizing the hyperparameters further.

To measure performance in the hyperparameter optimization, we use the silhouette score from Scikit-learn. We train on 75% of our data, stratified over true labels, and use the remaining 25% for testing. We measure performance on the test data using the adjusted Rand score [51].

[1] DBSCAN is the predecessor to HDBSCAN, which inspired SHDBSCAN.

4.3 Results

We take the mean of the several runs for each algorithm (100 runs for KMeans, DBSCAN, HDBSCAN, and SHDBSCAN and 10 runs for AffProp and SpecClustRBF due to their high computational cost) and plot these in Fig. 4. Vertical colored lines indicate the 95% experimental confidence interval on these individual scores except for AffProp and SpecClustRBF, where they indicate the 80% interval. In addition, we bootstrap on these scores (1000 trials, pulled with replacement) and mark the 95% experimental confidence interval on the average performance of each clusterer with black lines. SHDBSCAN's performance is noted with a red dot in the corresponding column. Values below 0 are ignored in this plot; these correspond to algorithms whose clustering resulted in an undefined silhouette score according to the Scikit-learn implementation, most often when all points are put in a single cluster.

In general, SHDBSCAN resulted in adjusted Rand scores relatively close to the other algorithms. It is in the top three performers in four of the twelve datasets used (2, 3, 5, and 6), in the bottom three performers for two datasets (1 and 10), and essentially indistinguishable from or comfortably on par with many of the other algorithms in the remaining six datasets (4, 7, 8, 9, 11, and 12). Although it does not particularly improve on clustering on these benchmark datasets, SHDBSCAN's clustering returned positive — if sometimes low — silhouette scores for every benchmark dataset we tested, whereas the other algorithms, most notably Spectral Clustering, sometimes did not result in positive scores.

5 Experiment 2: Discovery of Logic Gate Behavior by Clustering Behaviors

This experiment evaluates the efficacy of describing device function via the proposed behavior vectors in identifying devices with self-consistent behavior.

5.1 Data Generation: Synthetic 4-port Devices

First, we generate synthetic data that would mimic the data we would see in a fluid simulation. We consider only four-port fluidic logic gates with two inputs, one output, and one open "miscellaneous" port, as in Fig. 2.

Two-input logic gates have $2^2 = 4$ distinct possible input conditions [2]. With these four input states, we can define $2^4 = 16$ possible devices. Some of these devices have familiar behavior, like the AND gate; others are not particularly useful, like a constant "off" output. For each device, we:

1. Assign a true label,
2. Set the inlet velocity profiles,
3. Determine the corresponding outlet velocity profile,
4. Set the inlet and outlet pressure profiles, and
5. Enforce conservation of mass through the miscellaneous port.

5.1.1 Assigning a True Label

Consider the four possible inlet conditions: {0, 0}, {0, 1}, {1, 0}, and {1, 1}. For each of these conditions, the outlet can be either 0 or 1, as determined by the type of device. If we always refer to these inlet conditions in the same order, we can simplify a device's output to a four-digit binary number, consisting only of its outputs (0000, 0001, 0010, and so on). We can then convert this binary number into a decimal integer — between 0 and 15, inclusive — and use this number as the true label for each device. This compactly represents a device's type while also giving information about its output behavior.

Converting from the decimal representation to the output behavior is simply performing the process in reverse.

5.1.2 Setting Port Values

INLET VELOCITY For each device, we consider the four inlet conditions. If the condition is 1, we select a polynomial of order 0, 2, 4, or 6. If the polynomial order is not 0, we set its coefficients such that, at the edges of a port, the polynomial equals 0, while in the middle, the polynomial equals 1. We then multiply this polynomial by a random positive value (absolute value of a normally distributed value, $\mu = 10, \sigma = 1.5$). If the polynomial order is 0, we set a constant input velocity, distributed in the same way. These velocities are perpendicular to the port's orientation (*e.g.*, the left port has velocity purely in the positive x-direction, and the bottom port has velocity purely in the positive y-direction).

If the inlet condition is 0, we set the velocity to 0.

OUTLET VELOCITY For each inlet condition, we consider the appropriate outlet condition based on the true label of the device. If the outlet should be 1, we assume the outlet has a parabolic velocity profile and determine its maximum velocity as a random proportion (uniformly distributed, in $[0.7, 0.99]$) of the mean of the inflow maximum velocities. That this value is below 1 ensures that the miscellaneous port will have some mass flow and mimics frictional losses in the fluidic device. Similar to the inlet, the profile is perpendicular to the port's orientation (*e.g.*, since the outlet port is on the right, the velocity is purely in the positive x-direction).

If the outlet should be 0, we set the velocity to 0.

PRESSURES We set the first inlet pressure to a random constant value (uniformly distributed, in $[1.0, 100.0]$) and the second inlet to a random proportion of the first inlet's pressure (absolute value of a normally distributed value, $\mu = 1.0, \sigma = 0.2$).

We set the outlet and miscellaneous port pressures to a constant 0.

[2] Assuming inputs and outputs are either "on" or "off" and not "partially on."

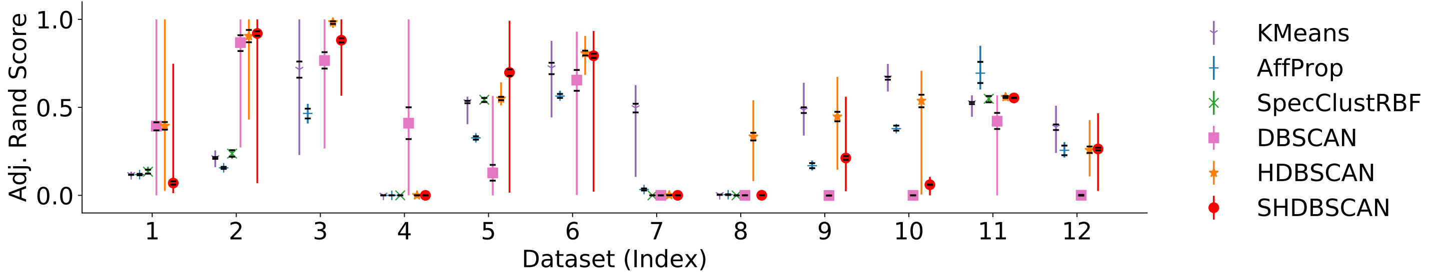


FIGURE 4: Performance spread of various algorithms on synthetic and benchmark datasets. SHDBSCAN is represented with a red dot. Algorithms with a performance below 0 are ignored in this plot. Dataset indices are referenced in Appendix D. Colored markers represent means of 10-100 train-test-split trials. Vertical bars represent the experimental confidence interval on adjusted Rand scores. We bootstrap on our dataset (1000 trials, pulled with replacement) and mark the 95% experimental confidence interval of average clusterer performance with black lines.

CONSERVATION OF MASS We subtract the outflow velocity profile from the sum of the inflow velocities to calculate the miscellaneous port profile. Since every port has the same width, this approximates conservation of mass in our device. This velocity will be perpendicular to the port direction.

We sample each device within the ports at the resolution set according to our heuristic in §3.2.1; here, this results in twenty equally spaced points along each port.

We linearly scale our behavior vectors as explained in §3.2.2. It is possible to scale the data differently — *e.g.*, the X- and Y-direction velocities as one larger group or by scaling the magnitudes of each vector rather than its components — but we do not explore these methods in this paper or expect significant differences from our results.

We append the behavior vectors of all four possible input states to form the behavior vector for a single device. For our data, this corresponds to:

1. number of ports = 4
2. number of dimensions in solution fields = velocity (x and y) + pressure = $2+1=3$
3. number of input states = $2^2=4$

giving a total of $4*3*4=48$ times the resolution number (as stated above, 20 in our data) elements in each unmodified behavior vector. Thus, each unmodified behavior vector has a length of $48*20=960$.

5.2 Baseline Methods: Sampling, Dimensionality Reduction, and Clustering

We generate behavior vectors using the methods from §3.2 on this data. We also compare PCA [52] ($n_components =$ 4, explaining at least 75% of the variance) and TSNE [53] ($n_components = 2$ and $perplexity = 40$)[3], both implemented in the Scikit-learn library, to reduce the dimensionality of our behavior vectors additionally, though at the cost of more abstract/less intuitive behavior vector terms and longer computational time.

We use the same clustering methods, cross validation, and hyperparameter optimization as in §4.2.

5.3 Results

Similar to above, we run our clustering algorithms on our preprocessed synthetic data and aggregate the data by dataset. We take the mean of the several runs for each algorithm (100 runs for KMeans and SHDBSCAN, 10 runs for DBSCAN and HDBSCAN, 2 runs for AffProp, and 1 run for SpecClustRBF, with the various algorithms' high computational costs limiting the number of feasible runs) and plot these in Fig. 5. Vertical colored lines indicate the 95% experimental confidence interval on these individual scores for KMeans and SHDBSCAN, the 80% interval for DBSCAN and HDBSCAN, and both points of AffProp. AffProp and SpecClustRBF were unable to run in a reasonable amount of time to generate multiple points to calculate confidence intervals. In addition, we bootstrap on these scores (1000 trials, pulled with replacement, with the exception of AffProp and SpecClustRBF) and mark the 95% experimental confidence interval on the average performance of each clusterer with black lines. SHDBSCAN's performance is again noted with a red dot in the corresponding column.

To measure performance, we calculate the adjusted Rand

[3]TSNE is originally intended for visualizing high-dimensional data [53]. In this vein, we set $n_components = 2$ for visualization. Although TSNE does not preserve distances or density, we decided to benchmark against it due to its prevalence as a method for dimensionality reduction. To choose the perplexity value, we test several values from 2 to 500. From our visualizations, perplexity values between 25 and 500 do not qualitatively affect the output. However, we do not quantitatively test this.

score for each algorithm by comparing the returned clustering and the true labels. These labels correspond to the type of device, and consequently the fluidic output behavior, as defined in §5.1. Thus, a high-performing algorithm would cluster all of the #0 devices together, all of the #1 devices together, all of the #2 devices together, etc. without clustering a #6 device with the #4s, for example. However, the algorithms do not return the type of device that constitutes each cluster.

From Fig. 5, we can see three main groups of performance. In the lowest group, columns 1, 2, 4, and 7 correspond to No Preprocessing, High Variance Filter, PCA, and High Variance Filter + PCA, respectively. These four preprocessing methods, regardless of clustering algorithm, have Adjusted Rand scores generally below 0.3. However, SHDBSCAN performs surprisingly well on average with No Preprocessing in column 1 and a High Variance Filter in column 2.

The middle performance group, from columns 3, 5, 6, 8, 10, 11, 13, and 14, correspond to a variety of preprocessing steps, referenced in Table 1. These eight methods generally fall near an Adjusted Rand score of 0.8, though the spread is larger, ranging from around 0.5 to nearly 1.0. Notably, SHDBSCAN performs well relative to the other clustering algorithms except in column 11 (PCA + TSNE).

The final group of performance are the high-performers. Columns 9, 12, 15, and 16, corresponding to Resolution Selection + PCA, High Variance Filter + Resolution Selection + PCA, Resolution Selection + PCA + TSNE, and High Variance Filter + Resolution Selection + PCA + TSNE, respectively, almost always had adjusted Rand scores of greater than 0.9, with the tightest spread of scores, though with a few outliers. In this group, SHDBSCAN performs well according to the Adjusted Rand score.

TABLE 1: Preprocessing steps used by column number, as seen in Fig. 5.

Column	High Variance	Resolution	PCA	TSNE
1				
2	X			
3		X		
4			X	
5				X
6	X	X		
7	X		X	
8	X			X
9		X	X	
10		X		X
11			X	X
12	X	X	X	
13	X	X		X
14	X		X	X
15		X	X	X
16	X	X	X	X

6 Discussion

From Experiment 1 and shown in Fig. 4, we note that SHDBSCAN does not perform better in many of the benchmark test cases — and sometimes performs significantly worse — compared to the benchmark algorithms. Our modifications to the HDBSCAN algorithm seemed fairly conservative, so this is somewhat surprising. Previous algorithms can already perform well on the benchmark datasets that we tested, so for SHDBSCAN to perform similarly at best shows that our modifications do not improve SHDBSCAN's performance on datasets similar to our benchmark data. Datasets 1, 4, 7, 8, and 10 show SHDBSCAN performing most poorly on an absolute scale; relative to the remaining algorithms, SHDBSCAN is relatively on par in Datasets 7 and 8, with the exception of KMeans and HDBSCAN, respectively. Five of these six datasets have data that is not distributed in some circular pattern in our visualization. The use of silhouette score in SHDBSCAN thus may cause such datasets not to have as highly rated clusterings. However, we made these modifications intending to improve performance on behavior vector clustering, which is tested in Experiment 2.

From Experiment 2, seen in Fig. 5, we can interpret how the different preprocessing methods affect clustering performance when combined with the various clustering algorithms. We expect the preprocessing methods to be effective in similar tasks, though with different data.

We notice that the low-performing group consists of the unprocessed and High Variance Filtered data, with and without PCA applied. This suggests that High Variance Filtering and PCA, individually or together, are not enough for clustering algorithms to make meaningful clusters out of behavior vector data for fluidic logic gates. However, the addition of either TSNE or the Resolution Selection preprocessing improves clustering performance. In fact, all of the high-performing methods contain both the Resolution Selection preprocessing as well as PCA; the addition of the High Variance Filter to those methods does not seem to affect performance.

With any preprocessing, SHDBSCAN is generally among the highest performers of the clustering algorithms and has the tightest average performance confidence interval. SHBSCAN

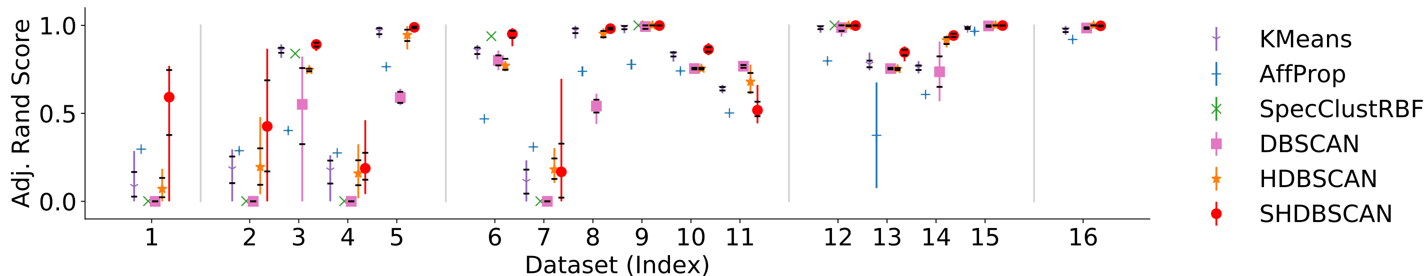


FIGURE 5: Performance spread of various clustering algorithms on preprocessed synthetic data. The first column set is the unprocessed data. The second column set uses one of the four preprocessing steps. The third column set uses two of the four preprocessing steps. The fourth column set uses three of the four preprocessing sets. The last column set uses all four preprocessing steps. The specific preprocessing used are shown in Table 1. SHDBSCAN is represented with a red dot. Colored markers represent means of 10-100 train-test-split trials. Vertical bars represent the experimental confidence interval on adjusted Rand scores. We bootstrap on our dataset (1000 trials, pulled with replacement) and mark the 95% experimental confidence interval of average clusterer performance with black lines (except for AffProp and SpecClustRBF).

also does surprisingly well on the unprocessed data. This may be due to the forced minimum cluster size matching the number of synthetic data points, though we do not study unbalanced cluster sizes further in this work. Since the majority of algorithms performed well in the high-performing group, there is little difference between SHDBSCAN and the other clustering algorithms. However, it does seem that SHDBSCAN, when combined with the Resolution Selection preprocessing and PCA dimensionality reduction, performs well in this type of problem.

As such, we find that the method of preprocessing the data is more important than the clustering method used. We find that using the Resolution Selection preprocessing, combined with PCA, gives the highest adjusted Rand scores, whereas the addition of the High Variance Filter or TSNE do not affect the outcome significantly. With the appropriate data preprocessing, all of the tested clustering algorithms perform at a similar level.

Despite the performance of SpecClustRBF and AffProp, the computational time needed was infeasible. In Experiment 2, we were only able to run 1-2 iterations of each, as opposed to the 10-100 iterations for the other four clustering algorithms. Based on wall-clock time, SpecClustRBF and AffProp took between 100 and 1000 times the amount of time needed for KMeans and SHDBSCAN, with DBSCAN and HDBSCAN falling somewhere in between. Although the wall-clock times were not formally tested or benchmarked, this experiment suggests SpecClustRBF and AffProp do not scale well to larger datasets.

In some cases, the silhouette score is undefined for a given clustering. For example, when all points are grouped in a single cluster, the silhouette score function is undefined; in this case, we return a negative value instead. An alternative situation arises when every point is put in its own cluster; again, if this is the case, we return a negative number. These are not included in our average or confidence interval calculations.

Although we use the example of fluidic logic gates in our work, we expect other representations or types of devices to perform similarly. We predict that the choice of behavior vector for a given application will have a stronger impact on the performance of all downstream algorithms. Identifying methods to define behavior vectors most "effectively," in whatever context it exists, is an avenue for future work. Choosing an appropriate behavior vector will likely improve the performance of all algorithms. We explore only a few basic methods of defining behavior vectors, but more work is needed.

7 CONCLUSION

Computational design methods show potential to discover novel and diverse designs that traditional optimization methods may miss. We explore the performance of several clustering algorithms and preprocessing heuristics in the discovery of synthesized fluidic logic gate devices, leading to the following two conclusions:

1. Our modifications on the HDBSCAN algorithm, which we call the SHDBSCAN algorithm, generally perform better via adjusted Rand scores in clustering behavior vectors than the clustering algorithms against which we benchmark. However, this effect is relatively minor compared to the effect of preprocessing.
2. Of the dimensionality reduction and preprocessing methods we test, the combination of the Resolution Selection preprocessing and PCA result in clusterings with the highest adjusted Rand scores. TSNE and the High Variance Filter have minor effects. The preprocessing effect overshadows the effect of the clustering algorithm selection.

This work explores a new avenue of computationally dis-

covering groups of devices from unlabelled data, even when the device's behavior is not known *a priori*. Exploring expanded design spaces computationally can uncover novel and diverse behaviors and devices that have been previously unexplored, even without the need for human guidance. This has the potential to invent families of devices that have been overlooked by human designers.

There are several directions in which to expand this work in the future. For example, applying these methods to different types of physics simulations can be explored to ensure the methods are as effective. In addition, methodologies to generate behavior vectors and their applications to other design problems provides an avenue for research — *e.g.*, how can we measure how well a behavior vector fits, and how can we define "better" behavior vectors?

ACKNOWLEDGMENT

We acknowledge the funding and support provided by DARPA through their Fundamentals of Design program (#HR0011-18-9-0009). The views, opinions, and/or findings contained in this article are those of the author and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

REFERENCES

[1] Vestad, T., Marr, D. W. M., and Munakata, T., 2004. "Flow resistance for microfluidic logic operations". *Applied Physics Letters,* **84**(25), pp. 5074–5075.

[2] Weaver, J., Melin, J., Stark, D., Quake, S., and Horowitz, M., 2010. "Static control logic for microfluidic devices using pressure-gain valves". *Nature Physics,* **6**, 01.

[3] Prakash, M., and Gershenfeld, N., 2007. "Microfluidic Bubble Logic". *Science,* **315**(5813), pp. 832–835.

[4] Jandieri, V., Khomeriki, R., and Erni, D., 2018. "Realization of true all-optical AND logic gate based on nonlinear coupled air-hole type photonic crystal waveguides". *Optics express,* **26**(16), pp. 19845–19853.

[5] Wang, L., Qian, K., Huang, Y., Jin, N., Lai, H., Zhang, T., Li, C., Zhang, C., Bi, X., Wu, D., et al., 2015. "SynBioLGDB: a resource for experimentally validated logic gates in synthetic biology". *Scientific reports,* **5**, pp. 80–90.

[6] Monz, T., Kim, K., Hänsel, W., Riebe, M., Villar, A., Schindler, P., Chwalla, M., Hennrich, M., and Blatt, R., 2009. "Realization of the quantum Toffoli gate with trapped ions". *Physical review letters,* **102**(4), p. 040501.

[7] Shi, X.-F., 2018. "Deutsch, Toffoli, and CNOT Gates via Rydberg Blockade of Neutral Atoms". *Physical Review Applied,* **9**(5), p. 051001.

[8] Neudeck, P. G., Meredith, R. D., Chen, L., Spry, D. J., Nakley, L. M., and Hunter, G. W., 2016. "Prolonged silicon carbide integrated circuit operation in Venus surface atmospheric conditions". *AIP Advances,* **6**(12), p. 125119.

[9] Rousseeuw, P. J., 1987. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". *Journal of Computational and Applied Mathematics,* **20**, pp. 53 – 65.

[10] Chiu, K., and Fuge, M., 2019. "Checking the Automated Construction of Finite Element Simulations from Dirichlet Boundary Conditions". *ASME International Design Engineering Technical Conferences*, Anaheim, CA, August 18-21, 2019.

[11] Fan, W., Gordon, M. D., and Pathak, P., 2004. "A Generic Ranking Function Discovery Framework by Genetic Programming for Information Retrieval". *Inf. Process. Manage.,* **40**(4), May, pp. 587–602.

[12] Mahdavi, M., Chehreghani, M. H., Abolhassani, H., and Forsati, R., 2008. "Novel meta-heuristic algorithms for clustering web documents". *Applied Mathematics and Computation,* **201**(1), pp. 441 – 451.

[13] Wang, J.-L., Chiou, J.-M., and Mueller, H.-G., 2016. "Review of Functional Data Analysis". *Annual Review of Statistics and Its Applications,* **3**, pp. 257–295.

[14] Serban, N., and Wasserman, L., 2005. "CATS: Clustering after Transformation and Smoothing". *Journal of the American Statistical Association,* **100**(471), pp. 990–999.

[15] Coffey, N., Hinde, J., and Holian, E., 2014. "Clustering longitudinal profiles using P-splines and mixed effects models applied to time-course gene expression data". *Computational Statistics & Data Analysis,* **71**, pp. 14 – 29.

[16] Heinzl, F., and Tutz, G., 2014. "Clustering in linear-mixed models with a group fused lasso penalty". *Biometrical Journal,* **56**(1), pp. 44–68.

[17] Bellman, R., 1957. *Dynamic programming*. Princeton University Press, Princeton, NY.

[18] Aloise, D., Deshpande, A., Hansen, P., and Popat, P., 2009. "NP-hardness of Euclidean sum-of-squares clustering". *Machine Learning,* **75**(2), May, pp. 245–248.

[19] Yin, M., Hu, Y., Yang, F., Li, X., and Gu, W., 2011. "A novel hybrid K-harmonic means and gravitational search algorithm approach for clustering". *Expert Systems with Applications,* **38**, 08, pp. 9319–9324.

[20] Xu, X., Lu, L., He, P., Pan, Z., and Chen, L., 2013. "Improving constrained clustering via swarm intelligence". *Neurocomputing,* **116**, pp. 317 – 325. Advanced Theory and Methodology in Intelligent Computing.

[21] Forsati, R., Keikha, A., and Shamsfard, M., 2015. "An improved bee colony optimization algorithm with an application to document clustering". *Neurocomputing,* **159**, pp. 9 – 26.

[22] Frey, B. J., and Dueck, D., 2007. "Clustering by Pass-

ing Messages Between Data Points". *Science,* **315**(5814), pp. 972–976.

[23] Shi, J., and Malik, J., 2000. "Normalized cuts and image segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **22(8)**.

[24] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X., 1996. "A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226–231.

[25] Jacques, J., and Preda, C., 2013. "Functional Data Clustering: A Survey". *Advances in Data Analysis and Classification,* **8**, 09, pp. 231–255.

[26] Damen, N. B., and Toh, C. A., 2019. "Investigating Information: A Qualitative Analysis of Expert Designers' Information Representation and Structuring Behaviors". *ASME International Design Engineering Technical Conferences*, Anaheim, CA, August 18-21, 2019.

[27] Arlitt, R. M., Nix, A. A., Sen, C., and Stone, R. B., 2016. "Discovery of Mental Metadata Used for Analogy Formation in Function-Based Design". *J. Mech. Des.,* **138**(10), October, p. 101110.

[28] Goucher-Lambert, K., Gyory, J. T., Kotovsky, K., and Cagan, J., 2019. "Computationally Derived Adaptive Inspirational Stimuli for Real-time Design Support During Concept Generation". *ASME International Design Engineering Technical Conferences*, Anaheim, CA, August 18-21, 2019.

[29] Kirjavainen, S., and Hölttä-Otto, K., 2019. "To Classify or Combine: The Effects of Idea Generation Mechanisms on the Novelty and Quantity of Ideas". *ASME International Design Engineering Technical Conferences*, Anaheim, CA, August 18-21, 2019.

[30] Zhang, C., Kwon, Y. P., Kramer, J., Kim, E., and Agogino, A. M., 2017. "Deep Learning for Design in Concept Clustering". *ASME International Design Engineering Technical Conferences*, Cleveland, OH, August, 2017.

[31] Zhang, C., Kwon, Y. P., Kramer, J., Kim, E., and Agogino, A. M., 2017. "Concept Clustering in Design Teams: A Comparison of Human and Machine Clustering". *J. Mech. Des.,* **139**(11), October, p. 111414.

[32] Camburn, B., He, Y., Raviselvam, S., Luo, J., and Wood, K., 2019. "Evaluating Crowdsourced Design Concepts with Machine Learning". *ASME International Design Engineering Technical Conferences*, Anaheim, CA, August 18-21, 2019.

[33] Gyory, J. T., Goucher-Lambert, K., Kotovsky, K., and Cagan, J., 2019. "Exploring the Application of Network Analytics in Characterizing a Conceptual Design Space". *Proceedings of the Design Society: International Conference on Engineering Design,* **1**(1), p. 19531962.

[34] Fu, K., Cagan, J., Kotovsky, K., and Wood, K., 2013. "Discovering Structure in Design Databases Through Functional and Surface Based Mapping". *J. Mech. Des.,* **135**(3), March, p. 031006.

[35] Mikes, A., Edmonds, K., Stone, R. B., and DuPont, B., 2020. "Optimizing an Algorithm for Data Mining a Design Repository to Automate Functional Modeling". *Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Information in Engineering Conference*, Virtual, Online, August 17-19, 2020.

[36] Shu, D., Cunningham, J., Stump, G., Miller, S. W., Yukish, M. A., Simpson, T. W., and Tucker, C. S., 2020. "3D Design Using Generative Adversarial Networks and Physics-Based Validation". *J. Mech. Des.,* **142**(7), July, p. 071701.

[37] Chen, W., and Ahmed, F., 2021. "PaDGAN: Learning to Generate High-Quality Novel Designs". *J. Mech. Des.,* **143**(3), March, p. 031703.

[38] Hooshmand, A., and Campbell, M. I., 2015. "Tensegrity Form-Finding Using Generative Design Synthesis Approach". *Proceedings of the ASME 2015 International Design Engineering Technical Conferences and Information in Engineering Conference*, Boston, MA, August 2-5, 2015.

[39] van Diepen, M., and Shea, K., 2019. "A Spatial Grammar Method for the Computational Design Synthesis of Virtual Soft Locomotion Robots". *J. Mech. Des.,* **141**(10), Oct, p. 101402.

[40] Schwarz, J., and Shea, K., 2019. "Machine Learning-Augmented Stochastic Search for the Automated Synthesis and Optimization of Cooling Channels". *Proceedings of the ASME 2019 International Design Engineering Technical Conferences and Information in Engineering Conference*, Anaheim, CA, August 18-21, 2019.

[41] Rafibakhsh, N., Huang, W., and Campbell, M. I., 2018. "Automatic Detection of Fasteners From Tessellated Mechanical Assembly Models". *J. Comput. Inf. Sci. Eng.,* **18**(1), March, p. 011005.

[42] Massoni, B., and Campbell, M. I., 2020. "Automated Decomposition of Complex Parts for Manufacturing with Advanced Joining Processes". *J. Manuf. Sci. Eng.,* **142**(6), June, p. 061002.

[43] Dering, M. L., and Tucker, C. S., 2013. "A Convolutional Neural Network Model for Predicting a Product's Function, Given Its Form". *J. Mech. Des.,* **135**(3), March, p. 031006.

[44] Nandy, A., Dong, A., and Goucher-Lambert, K., 2020. "A Comparison of Vector and Network-Based Measures for Assessing Design Similarity". *Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Information in Engineering Conference*, Virtual, Online, August 17-19, 2020.

[45] Campello, R. J. G. B., Moulavi, D., and Sander, J., 2013. "Density-Based Clustering Based on Hierarchical Density Estimates". *Advances in Knowledge Discovery and Data Mining*, pp. 160–172.

[46] Prim, R. C., 1957. "Shortest connection networks and some

generalizations". *The Bell System Technical Journal,* **36**(6), pp. 1389–1401.
[47] Dua, D., and Graff, C., 2017. UCI Machine Learning Repository.
[48] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., 2011. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research,* **12**, pp. 2825–2830.
[49] McInnes, L., Healy, J., and Astels, S., 2017. "HDBSCAN: Hierarchical density based clustering". *The Journal of Open Source Software,* **2**(11), March, 2017.
[50] Nogueira, F., 2014–. Bayesian Optimization: Open source constrained global optimization tool for Python.
[51] Hubert, L., and Arabie, P., 1985. "Comparing partitions". *Journal of Classification,* **2**, pp. 193–218.
[52] Tipping, M. E., and Bishop, C. M., 1999. "Probabilistic principal component analysis". *Journal of the Royal Statistical Society: Series B (Statistical Methodology),* **61(3)**, pp. 611–622.
[53] van der Maaten, L., and Hinton, G., 2008. "Visualizing Data using t-SNE". *Journal of Machine Learning Research,* **9**, pp. 2579–2605.
[54] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E., 2020. "Array programming with NumPy". *Nature,* **585**(7825), Sept., pp. 357–362.

A Sample Calculations for Resolution Selection on Example Behavior Vectors

We work through two examples of the resolution preprocessing step from §3.2.2: one where the behavior vectors are not shortened ($m^* = 1$), and one where they are ($m^* = 2$).

A.1 Example 1: $m^* = 1$

Consider the following 3 behavior vectors.

{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
{0.6, 0.4, 0.2, 0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 0.8}
{0.9, 0.8, 0.7, 0.7, 0.6, 0.7, 0.0, 0.0, 0.3, 0.4, 0.3}

After taking differences between adjacent points, we get:

{0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1}
{0.2, 0.2, 0.2, 0.1, 0.2, 0.2, 0.2, 0.2, 0.1, 0.2}
{0.1, 0.1, 0.0, 0.1, 0.1, 0.7, 0.0, 0.3, 0.1, 0.1}

Taking the maximum of each difference vector, we get:

0.1
0.2
0.7

Reciprocating these values, we get:

10
5
~ 1.43

Taking the floor of these values, we get:

10
5
1

These values correspond to the m values for our three example behavior vectors. The minimum of these is m^*. For this example, we would then take every (1-th) point to form the "shortened" behavior vectors.

Thus, the behavior vectors after this preprocessing are:

{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
{0.6, 0.4, 0.2, 0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 0.8}
{0.9, 0.8, 0.7, 0.7, 0.6, 0.7, 0.0, 0.0, 0.3, 0.4, 0.3}

A.2 Example 2: $m^* = 2$

Consider this second example with the following 3 behavior vectors of length 11. They are similar to the previous example with a minor change in the third vector.

{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
{0.6, 0.4, 0.2, 0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 0.8}
{0.9, 0.8, 0.7, 0.7, 0.6, 0.7, 0.2, 0.0, 0.3, 0.4, 0.3}

After taking differences between adjacent points, we get:

{0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1}
{0.2, 0.2, 0.2, 0.1, 0.2, 0.2, 0.2, 0.2, 0.1, 0.2}
{0.1, 0.1, 0.0, 0.1, 0.1, 0.5, 0.2, 0.3, 0.1, 0.1}

Taking the maximum of each difference vector, reciprocating these values, and taking the floor, we get:

10
5
2

For this example, $m^* = 2$, so we take every 2nd point to form the shortened behavior vectors.

Thus, the behavior vectors after this preprocessing are:

{0.0, 0.2, 0.4, 0.6, 0.8, 1.0}
{0.6, 0.2, 0.1, 0.5, 0.9, 0.8}
{0.9, 0.7, 0.6, 0.0, 0.3, 0.3}

B Exploration of Different Variance Cutoff Values for Synthetic Fluidic Logic Gates

We applied the High Variance Filter with cutoff values over the range [0, 1], in intervals of 0.001, and plotted this against the number of points in the resulting behavior vector. In Fig. 6, we see two plateaus: the first between 0.015 and 0.035, and the second above 0.04. These correspond to our expectations of low- and high-variance points, respectively, as mentioned in §3.2.3.

The second — and more generally, last — plateau can be interpreted as the trailing off of extremely high-variance points; these are expected to be the points we want to keep in any case, so we do not want to choose a cutoff in this region. Considering the first plateau, we choose a variance cutoff in this range, and as there is not a significantly large difference between cutoffs of 0.015 and 0.035, we choose a value of 0.02.

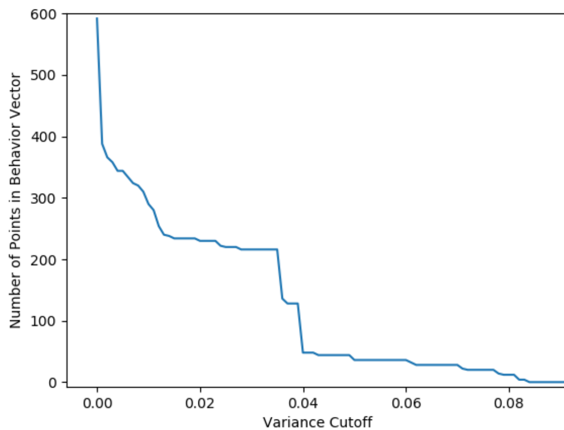


FIGURE 6: Length of behavior vectors after applying the High Variance Filter. Cutoff values above 0.09 are not shown as all behavior vectors are empty.

C Hyperparameter Ranges for Clustering Algorithms

TABLE 2: Hyperparameter names (and corresponding code nicknames), types, and ranges used in Bayesian optimization. Bayesian optimization is allowed to run 5 + 30 times the number of search parameters; *e.g.*, 35 iterations for KMeans, and 95 iterations for DBSCAN.

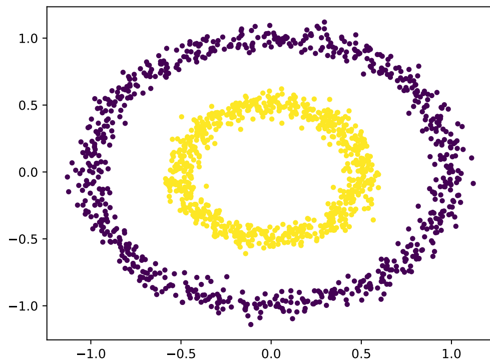
	Parameter 1				Parameter 2				Parameter 3			
Algorithm	Name	Type	Lower Bound	Upper Bound	Name	Type	Lower Bound	Upper Bound	Name	Type	Lower Bound	Upper Bound
KMeans	Number of Clusters (n_clusters)	int	2	20	-	-	-	-	-	-	-	-
DBSCAN	Epsilon (eps)	float	1e-5	5	Minimum Samples (min_samples)	int	2	100	Leaf Size (leaf_size)	int	2	20
HDBSCAN	Minimum Cluster Size (min_cluster_size)	int	2	100	Minimum Samples (min_samples)	int	2	100	Cluster Selection Epsilon (cluster_selection_epsilon)	float	1e-5	5
Affinity Propagation	Damping (damping)	float	0.5	1.0 - 1e-9	-	-	-	-	-	-	-	-
Spectral Clustering	Affinity (affinity)	string	RBF	-	Number of Clusters (n_clusters)	int	2	20	Gamma (gamma)	float	1e-5	5
SHDBSCAN	-	-	-	-	-	-	-	-	-	-	-	-

D Datasets used in Experiment 1

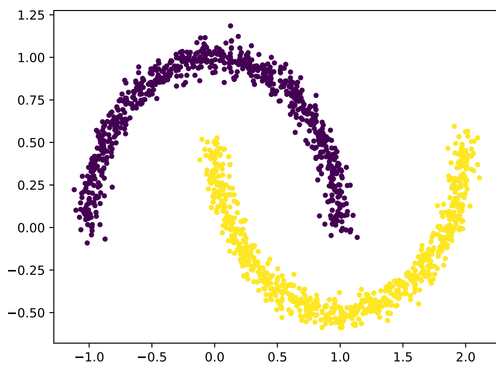
The datasets for Experiment 1 can be broken into two subgroups: synthetically generated and imported experimental data.

For the synthetically generated data, we use Scikit-learn's built-in "datasets" functions for generating classification data. We set the number of samples in each dataset to 1500 and initialize the internal random state to 170. True labels are colored. These are Figs. 7, 8, 9, and 10.

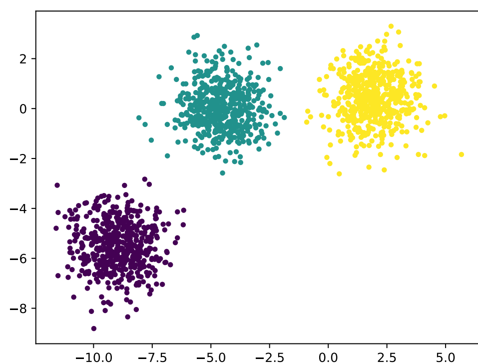
We import well-known datasets from the UCI ML Repository. We use TSNE, with default parameters as implemented in Scikit-learn, for visualization. These are Figs. 9c, 10a, 10b, and 10c.



(a) Index 1: Noisy circles. Generated with Scikit-learn's *make_circles* function. Additional parameters: $factor = 0.5$ and $noise = 0.05$

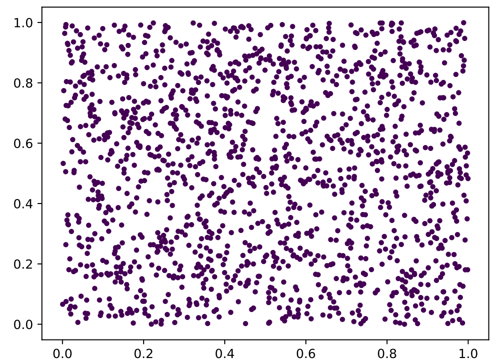


(b) Index 2: Noisy moons. Generated with Scikit-learn's *make_moons* function. Additional parameters: $noise = 0.05$

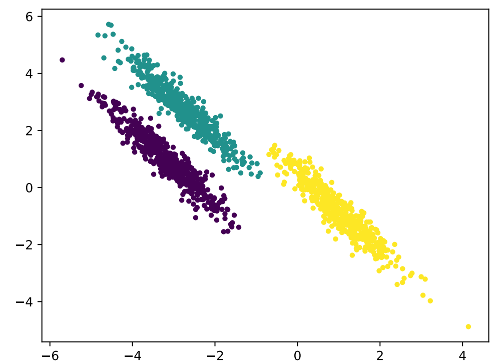


(c) Index 3: Blobs. Generated with Scikit-learn's *make_blobs* function. Additional parameters: none

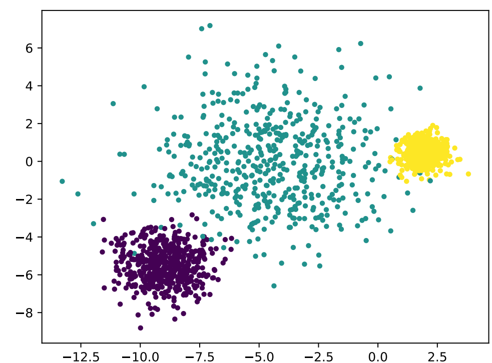
FIGURE 7: Datasets 1 to 3 used in Exp. 1



(a) Index 4: No structure. Generated with NumPy's [54] *random.rand* function in 2 dimensions.

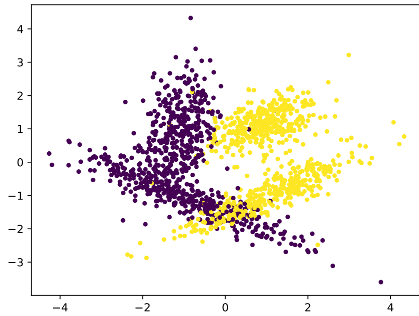


(b) Index 5: Anisotropically distributed blobs. Generated by taking the blobs dataset and applying a $[[0.6, -0.6], [-0.4, 0.8]]$ transformation.

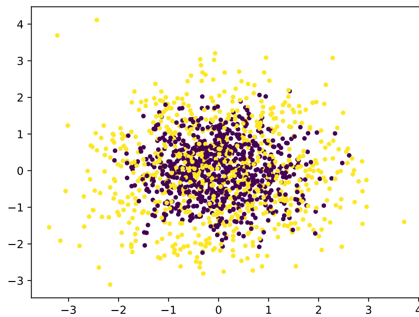


(c) Index 6: Blobs with varied variances. Generated with Scikit-learn's *make_blobs* function. Additional parameters: $cluster_std = [1.0, 2.5, 0.5]$

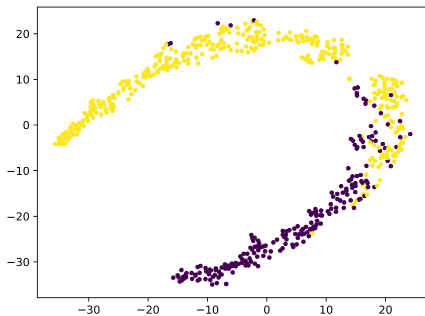
FIGURE 8: Datasets 4 to 6 used in Exp. 1



(a) Index 7: PCA representation of the classification dataset. Dataset is without PCA. Generated with the *make_classification* function from Scikit-learn. Additional parameters: none

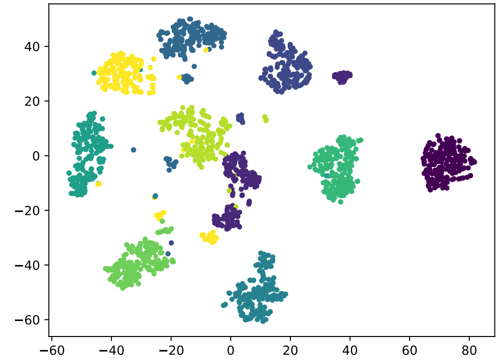


(b) Index 8: PCA representation of the hastie dataset. Dataset is without PCA. Generated using the *make_hastie_10_2* function from Scikit-learn. Adjusted slightly so all labels are non-negative integers. Additional parameters: none

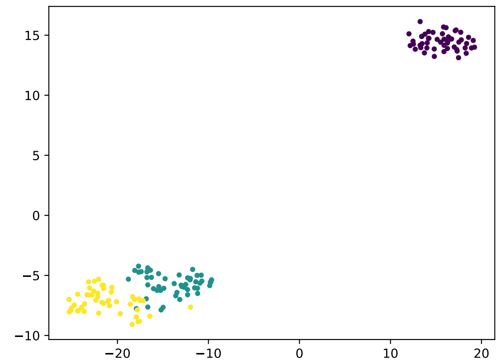


(c) Index 9: TSNE representation of the Breast Cancer Wisconsin dataset. Dataset is without TSNE.

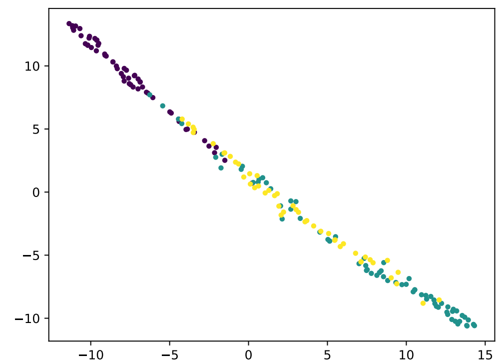
FIGURE 9: Datasets 7 to 9 used in Exp. 1



(a) Index 10: TSNE representation of the Handwritten Digits dataset. Dataset is without TSNE.



(b) Index 11: TSNE representation of the Iris dataset. Dataset is without TSNE.



(c) Index 12: TSNE representation of the Wine Recognition dataset. Dataset is without TSNE.

FIGURE 10: Datasets 10 to 12 used in Exp. 1