Ant Colony Stream Clustering: A Fast Density Clustering Algorithm for Dynamic Data Streams

Conor Fahy, Shengxiang Yang, *Senior Member, IEEE*, and Mario Gongora

***Abstract*—A data stream is a continuously arriving sequence of data and clustering data streams requires additional considerations to traditional clustering. A stream is potentially unbounded, data points arrive on-line and each data point can be examined only once. This imposes limitations on available memory and processing time. Furthermore, streams can be noisy and the number of clusters in the data and their statistical properties can change over time. This paper presents an on-line, bio-inspired approach to clustering dynamic data streams. The proposed Ant-Colony Stream Clustering (ACSC) algorithm is a density based clustering algorithm, whereby clusters are identified as high-density areas of the feature space separated by low-density areas. ACSC identifies clusters as groups of micro-clusters. The tumbling window model is used to read a stream and rough clusters are incrementally formed during a single pass of a window. A stochastic method is employed to find these rough clusters, this is shown to significantly speed the algorithm with only a minor cost to performance, as compared to a deterministic approach. The rough clusters are then refined using a method inspired by the observed sorting behaviour of ants. Ants pick-up and drop items based on the similarity with the surrounding items. Artificial ants sort clusters by probabilistically picking and dropping micro-clusters based on local density and local similarity. Clusters are summarised using their constituent micro-clusters and these summary statistics are stored offline. Experimental results show that the clustering quality of ACSC is scalable, robust to noise and favourable to leading ant-clustering and stream-clustering algorithms. It also requires fewer parameters and less computational time.**

***Index Terms*—Data stream clustering, density clustering, concept drift, concept evolution, ant colony**

I. Introduction

MINING data streams brings additional problems to traditional data mining. A data stream is a potentially unbounded sequence of data and in dynamic environments the properties of this data can change over time in unforeseen ways. As a stream progresses, the performance of traditional classifiers and predictive models can degrade as the characteristics of the target objects change. This change can be gradual, known as concept-drift, sudden as concept-shift, or in the form of concept-evolution when entirely new classes appear in the stream. Detecting this change is a challenge, as is reacting and adapting to the change. This challenge is compounded by the scarcity-of-labels problem, whereby in a streaming environment, newly arriving data is both expensive and time-consuming to label. Unsupervised learning techniques, such as clustering, can potentially be used to mitigate this labelling problem and also as a change detection mechanism alongside traditional classifiers and predictive models.

Clustering in a data stream requires additional considerations to traditional clustering. When dealing with a continuous sequence of information, it is only possible to examine the data once. Clustering needs to be performed quickly to prevent bottlenecks and potential loss of data. A stream can be potentially infinite but only a limited amount of memory is available, necessitating the summarisation of identified clusters in a meaningful way. The nature of an evolving stream implies that clusters can drift, new clusters can appear, or clusters can disappear and reappear cyclically. Therefore, it is difficult to know a priori how many clusters are present in the stream. Although traditional partitioning clustering techniques, such as *k-means* and its variants, have been successfully applied to data streaming, they have the drawback of requiring k to be specified a priori. Density based clustering, a form of hierarchical clustering, overcomes this limitation.

Density based clustering defines clusters as high-density areas of the feature space separated by areas of low density. It can identify arbitrarily shaped clusters, is robust to outliers and, crucially, does not require the number of clusters to be known a priori. In our proposed algorithm, dense areas are described using micro-clusters: n-dimensional spheres with centre c and radius r. Micro-clusters have a maximum radius ϵ where $r \leq \epsilon$. A data point is assigned to a micro-cluster if the point falls within its radius. The set of micro-clusters that are connected form the macro-cluster. Generally, there are more micro-clusters than there are actual clusters but significantly fewer micro-clusters than there are data points. This serves a dual purpose both as the clustering mechanism and as a summarisation technique because a number of local data points can be represented by a single micro-cluster. Clusters identified by the algorithm are summarized by their constituent micro-clusters and these summaries are stored off-line for evaluation by the user. This has two advantages: 1) information about clusters can be stored in a fraction of the space and 2) representative micro-clusters are potentially easier to evaluate than the entire set of individual data points assigned to a cluster.

A density based approach to stream clustering can address the problem of a shifting number of non-stationary clusters and provides a method to summarize these clusters, and we propose a sampling method to address the speed requirement

Manuscript received November 2, 2016; revised August 21, 2017; December 3, 2017 and March 7 2018; accepted March 27, 2018. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under Grant EP/K001310/1 (*Corresponding author: Shengxiang Yang*).

The authors are with the Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, U.K. (email: {conor.fahy, syang, mgongora}@dmu.ac.uk).

of data stream clustering. A point's similarity with a cluster is evaluated using a sample taken from the cluster. The stochastic sampling method replaces the traditional, exhaustive search for each point's appropriate micro-cluster, and subsequently the nearest neighbour of each micro-cluster. Rough clusters are incrementally created in a single pass of the data. The first point seeds the first cluster, subsequent points are assigned to an existing cluster or, if too dissimilar, seed a new cluster. Only after every point has been assigned to its respective cluster are micro-clusters created. Each point is converted to a micro-cluster and these micro-clusters attempt to merge with others in the *same* cluster only. The merging operation is expensive, and attempting to merge at this stage reduces the number of failed merging attempts.

After this single-pass of the data, the discovered clusters are often rough and too many. These clusters are refined using an ant-inspired sorting method. This method (i.e., the "pick-and-drop" method) is modelled based on the behavior of certain species of ant which cluster corpses into "cemeteries" or sort their larvae into piles. The idea is that isolated items should be picked up and then dropped at other locations where similar items are present. Sorting Ants are assigned to each cluster and they attempt to refine the initial clusters by probabilistically picking micro-clusters and dropping them in more suitable clusters. The probabilistic operations are biased toward the dissolution of smaller clusters and their contents moved to similar, larger clusters. Intuitively, it's better to have all points of class X in one cluster rather than distributed in a number of smaller clusters.

In other density based streaming algorithms that use micro-clusters, e.g., *DenStream* [7] and its variants, micro-clusters are defined by two parameters; the maximum radius ϵ and $minPoints$ which determines the minimum number of data points within ϵ for the micro-cluster to be considered dense. In ACSC, each point is initially treated as its own micro-cluster and so the $minPoint$ parameter is effectively 1 and therefore not required. This removes the complication of defining micro-clusters as either *core*, *potential* or *outlier* as each micro-cluster is treated equally. To further simplify we merge the concepts of *density-reachable*, *directly density-reachable*, and *density-connected* into one concept *density-reachable*, which determines if two micro-clusters are connected and should be considered part of the same cluster. ACSC assigns points to a cluster *before* creating and merging micro-clusters so when a new micro-cluster is directly density-reachable to any in the cluster, it is density-reachable and density-connected to all in the cluster. This simplifies the algorithm, reduces the overall complexity and allows for effective sampling. The main features of ACSC can be summarized as follows:

- The two phases of summarisation and clustering are combined into a single on-line phase.
- Micro-clusters are defined using a single parameter ϵ, and ACSC requires just three parameters overall.
- Clusters are formed using a single concept of density and outliers are identified as clusters containing a single point.
- Sorting operations are performed locally, through sampling from each cluster. This reduces the computational time and scales linearly to larger dimensionality and increasing numbers of clusters.

The rest of this paper is organized as follows. Section II presents related work in this area. Our proposed ACSC is described in Section III. Section IV presents our experimental study based on several real and synthetic data streams. Section V concludes this paper.

II. Related Work

The work by Aggarwal *et al.* [1] was the first attempt to address one of the fundamental problems in dealing with streams, the problem of being unable to revisit evolving data. The authors suggested that a stream clustering algorithm should consist of two components: an online component and an offline component. Data arriving online should be summarized and the offline component should perform clustering on the summarized data. Their algorithm, *CluStream*, introduced the concept of micro-clusters as a method to summarize data. Micro-clusters are a temporal extension of the cluster-feature-vector proposed in [45]. In CluStream, only a certain number of micro-clusters can be stored in memory at any one time so when a new micro-cluster is formed, two existing micro-clusters must be merged or one deleted. The offline clustering of the micro-clusters is based on the *k-means* algorithm [20].

Partitional clustering algorithms have been extended for single-pass and stream clustering. In [33] a general approach is proposed to enable traditional soft partitional clustering algorithms to deal with streaming data. The data stream is split into chunks and each chunk is partitioned into a set of cluster centroids. The centroids are weighted with the amount of samples they represent and in order to maintain the history of the stream, previously identified centroids are added to the newly arriving chunk of data to be clustered. In [17] a constant-factor approximation algorithm for a K-Median approach to data streaming is described. These algorithms offer a fast, accurate single pass clustering of data but could be sensitive to changes in the underlying distribution or a shifting number of natural clusters.

Density based approaches [12] do not suffer from these limitations. Clusters are identified as areas of high density in the feature space and so the number of clusters does not need to be specified, noise and outlier points are easier to identify and clusters of any shape can be found. Density based clustering has been extended for streams by adopting the aforementioned, two-part, online and offline framework [7], [41], [43].

MR-Stream [43] partitions the search space into cells and uses a tree structure to store the space partitioning. Newly arriving data points are assigned to a cell and the tree structure is updated. The off-line clustering is performed on the summarized tree.D-Stream introduced in [41] and extended in [8] also partitions the search space into discrete grid sections on which data are mapped. A decay factor is used to give higher importance to recent data. The authors introduced a technique to detect and remove sparse grid sections, which improves the space and time efficiency and improves on previous grid based clustering algorithms.

DenStream [7] extends the concept of micro clusters introduced in CluStream by adding a temporal aspect. It uses

the time-dampened window model to assign higher weight to recent data. Data points are summarized on-line as micro-clusters and when a clustering request is made by a user, these micro-clusters are clustered off-line using a traditional density-based clustering algorithm [12]. However, as observed in [15], [43], the offline clustering phase is computationally expensive and, furthermore, it is only executed when a clustering request is made by a user, so there is potentially a trade-off between frequent requests to discover changes in the stream and infrequent requests in order to reduce computational overheads.

To overcome this, the two phases of DenStream were merged into a single online phase in *FlockStream* [15]. Flockstream was inspired by the flocking behavior of birds as proposed in Reynolds' Boids algorithm [36]. It uses a decentralised, self-organising strategy to group similar micro-clusters. FlockStream has been shown to require substantially fewer pair-wise distance comparisons than DenStream while achieving similar cluster purity. FlockStream adopts the concepts of time-weighted *core* micro-clusters and *non-core* micro-clusters introduced in DenStream. These concepts are also employed in DEC [3] to identify clusters in real-time streams upon which fuzzy models are developed.

The off-line clustering phase used by DenStream is an extension of the DBSCAN algorithm [12], which groups data in terms of core-points, reachable points, and outliers. A point p is considered core if there are at least $minPoints$ within a distance ϵ of p. Point p forms a cluster with all points that are reachable to it and any points which are not are considered to be outliers. Various extensions to DBSCAN have been proposed, including methods for parallelisation [29], parameter estimation [13], and increased accuracy [31]. A comprehensive review of all the variants is given in [35].

A particularly relevant extension of DBSCAN is the PACA-DBSCAN algorithm [23]. PACA-DBSCAN uses the pick-and-drop model of ant clustering introduced in [9] to initially partition the data before the DBSCAN algorithm is applied. The pick-and-drop model proposed by Deneubourg *et al.* was extended [30] for data analysis by introducing a similarity measure between data objects. In this model, each *n*-dimensional data point is associated with a 2-dimensional (2D) point in a 2D space. Initially, these points are distributed randomly. Artificial ants move around and probabilistically pick-and-drop the 2D data objects based on local density in the 2D space and similarity of the data in the *n*-dimensional feature space. Clusters emerge as a consequence of these simple, local rules. This pick-and-drop model has been extended [18], [6], [19]. However, this model requires many iterations to cluster the data and the final clustering solution is just a spatial embedding of the data in a 2D space and a further processing step is often required. Along with hard clustering methods, fuzzy approaches to ant clustering have been proposed [24]. Once initial heaps have been formed by the ants, the centroids are refined by the Fuzzy C Means algorithm [4].

A variation on the Leader Ant algorithm was proposed in *AntClust* [27]. Data objects are associated with ants and ants are assigned an "odour". Ants sharing the same odour form nests. This model was extended in [32] for streams and is one of the few examples in the literature of ant-inspired stream clustering algorithms. Ants move from nest to nest along pheromone trails and find their most suitable (closest in terms of the Euclidean distance). The algorithm computes the clusters using k-means, which limits its suitability for a streaming environment.

Along with pick-and-drop and chemical coordination, other properties of social insects, such as self assembly [2] and stimergy [14], have been used as inspiration for clustering models, and there is a large body of research on clustering with Ant Colony Optimisation (ACO) [11] whereby the clustering problem is viewed as an optimisation problem [25], [37], [38]. Other bio-inspired stochastic stream-clustering algorithms include extensions on the Neural Gas algorithm G-Stream [16] and Kohen's Self Organising Map [10].

III. Proposed Ant-Colony Stream Clustering (ACSC) Algorithm

ACSC employs the tumbling window model [28] when dealing with data streams. A tumbling window is a type of sliding window where, at each iteration, a fixed size non-overlapping chunk of data is considered. In each window, ACSC identifies clusters as a group of micro-clusters, and a micro-cluster is a set of neighbouring points within a certain radius; the ϵ-neighbourhood determines this radius.

A micro-cluster containing N points $\{\vec{X}_i\}$, $i = \{1, ..., N\}$, is described using three components: the number of data points the micro-cluster contains (N), the Linear Sum (LS) of each dimension (i.e., $\sum_{i=1}^{N} \vec{X}_i$), the Squared Sum (SS) of each dimension (i.e., $\sum_{i=1}^{N} \vec{X}_i^2$). LS and SS are d-dimensional arrays, where d is the number of dimensions in a point. From these, the radius r and centre c of the micro-cluster can be determined [1]:

$$c = \frac{LS}{N} \tag{1}$$

$$r = \sqrt{\frac{SS}{N} - \left(\frac{LS}{N}\right)^2} \tag{2}$$

A micro-cluster can also contain a temporal variable, but this is not necessary in the tumbling window model. LS and SS have the properties of additivity and increment-ability, which allow micro-clusters to absorb new data points and to merge with other micro-clusters. A micro-cluster m can absorb point p if, after updating the LS and SS of m with p, $radius(m) \leq \epsilon$. Similarly, two micro-clusters m_i and m_j can attempt to merge into a single micro-cluster m_k as follows:

$$m_k = (N_i + N_j, \vec{LS}_i + \vec{LS}_j, SS_i + SS_j) \tag{3}$$

If $radius(m_k) \leq \epsilon$, the clusters merge; otherwise, the merging operation fails. The pseudo-code for this process is outlined in Algorithm 1.

Micro-clusters m_i and m_j are said *density reachable* if:

$$dist(c_{m_i}, c_{m_j}) \leq \epsilon, \tag{4}$$

where c_{m_i} and c_{m_j} are the centres of micro-clusters m_i and m_j, respectively.

Algorithm 1 Merge Operation

Input : ϵ-neighbourhood, 2 micro-clusters; a and b
Output : Merged micro-cluster *iff* operation successful

Create new, empty micro-cluster c
Initialise $c := a$
Add b to c (Eq. (3))
r := radius of c (Eq. (2))
if $(r \leq \epsilon)$ **then** merge successful
    Delete a and b
    Return c
else
    Delete c
    Return false

The solution provided by ACSC is a set of clusters containing density-reachable micro-clusters. ACSC works in two steps: 1) Rough clusters are identified in a single-pass of the window and 2) these rough clusters are refined and their summary statistics are stored off-line.

A. Find Clusters

ACSC employs the tumbling window model when dealing with data streams. At each iteration, a fixed size non-overlapping chunk of data is considered. So, at the beginning of this step, there will be $WindowSize$ points. In a single-pass of the window, clusters are incrementally formed. The first point seeds the first cluster. Subsequent points are assigned to an existing cluster or used to seed a new cluster. Point p's suitability with cluster c is evaluated using the Euclidean distance from p with a sample $nSamples$ taken from c. The suitability of point p with cluster c is estimated as follows:

$$Suitability(p, c) = \frac{\sum_{j=1}^{min(nSamples, n)} dist(p, c_j)}{min(nSamples, n)}, \quad (5)$$

where n is the number of points that are present in the cluster. Each cluster is evaluated and the point is assigned to the most suitable one *provided* the suitability is equal to or below ϵ; if not, the point seeds a new cluster. The parameter ϵ determines the maximum radius for a micro-cluster in the subsequent step and also serves as the minimum suitability measure in this step.

As we estimate each point's suitability with each cluster, we $record$ each suitability. This is useful information, especially when afforded just a single pass of the data. Upon joining (or establishing) a cluster, we update the similarity information between the selected cluster and its neighbouring clusters. The similarity between cluster a and cluster b is the average of each point p in cluster a's suitability (Eq. (5)) with cluster b:

$$Similarity(a, b) = \frac{1}{n} \sum_{i=1}^{n} Suitability(a_i, b) \quad (6)$$

The similarity to each neighbouring cluster is a rolling average updated whenever a new point is assigned to the cluster. Although comparable, $Similarity(a, b) \neq Similarity(b, a)$. This phase of the algorithm is outlined in Algorithm 2.

Algorithm 2 Find Clusters

Input : Window
Output : Clusters, Cluster Similarity

while <Window> **do**
    for <each data point> **do**
        if <clusters> **then**
            Find best cluster (Eq. (5))
            Add point to cluster
            Update cluster similarity (Eq. (6))
        else if <No clusters || No suitable cluster> **then**
            Create new cluster
            Add point to cluster
            Update cluster similarity (Eq. (6))
return Clusters, Cluster Similarity

B. Sort Clusters

The previous step discovers clusters in a single-pass of the window. The clusters identified at this stage are often rough, impure and too-many. In this step, micro-clusters are created, merged, and inter-cluster sorting is performed.

Initially, each d-dimensional point p in each cluster is treated as its own micro-cluster m. This micro-cluster will have a radius of 0 and a centre of p. Formally, we have:

$$\begin{aligned} m.N &= 1 \\ m.LS_i &= p_i, i = \{1, \cdots, d\} \\ m.SS_i &= {p_i}^2, i = \{1, \cdots, d\} \end{aligned} \quad (7)$$

where p_i is the i^{th} dimension of point p.

Before sorting, each micro-cluster attempts to merge with other micro-clusters in the same cluster. The merging operation is performed by comparing each micro-cluster with every other in the same cluster. If, after summing their constituent parts (Eq. (3)), the radius is less than or equal to ϵ, the merging operation is a success. Merging at this step ensures that only neighbouring micro-clusters attempt to merge and avoids the unnecessary computation of comparing micro-clusters in different dense areas. Another advantage is that, during the sorting phase, n points represented by a micro-cluster can be moved in a single operation, speeding the sorting process and reducing the number of pairwise comparisons.

'Sorting Ants' are created and one is assigned to each cluster. Each Sorting Ant is native to its own cluster. Sorting Ants probabilistically decide to pick-up a micro-cluster from their cluster. A micro-cluster m is chosen at random from cluster k and is iteratively compared with $nSamples$ micro-clusters in the same cluster. The Euclidean distance from the centre of m to each of the selected micro-clusters is calculated and if both are density-reachable (Eq. (4)), then a *reachable* count is incremented. The probability of a pick-up is calculated as follows:

$$P_{pick} = 1 - \frac{reachable}{nSamples} \quad (8)$$

It is important to note that if the number of micro-clusters n in cluster c is fewer than $nSamples$, then only n comparisons

Algorithm 3 Sort Clusters

Input : Initial Clusters, Cluster Similarity
Output : Sorted Clusters

```
Create micro-clusters (Eq. 7)
Merge micro-clusters in each cluster (Algorithm 1)
Assign Sorting Ant to each cluster
while <!Stop Condition> do
    for <each ant> do
        if <!Sleeping> then
            Probabilistically pick-up (Eq. (8))
            if <Carrying> then
                Move to most similar cluster
                Probabilistically drop (Eq. (8))
        Update similarity information (Eq. (6))
        Update sleepCounter
return Clusters
```

are made. However, P_{pick} is still calculated using $nSamples$. This ensures a higher probability of a pick-up in clusters containing fewer micro-clusters. This leads to the dissolution of smaller clusters and their incorporation into larger, similar clusters.

If a micro-cluster is successfully picked-up, the Boolean variable *carrying* is true and the Sorting Ant moves to a neighbouring cluster and attempts to drop it.

Sorting Ants move to the most similar cluster (using the similarity information from the first step) ensuring that they do not attempt to drop micro-clusters in clusters that are dissimilar to their own. A sorting ant attempts to drop its micro-cluster in the new cluster based on the inverse of Eq. (8). If the dropping operation is successful, the micro-cluster is moved to the new cluster; otherwise, the micro-cluster remains in its original cluster. The ant returns to its native cluster and updates the similarity information between the two clusters with the latest suitability score, see Eq. (5).

Each Sorting Ant continues to attempt sorting until either the cluster is empty (all of its contents have been moved to another cluster) or the Sorting Ant is 'asleep'. Each Sorting Ant has a counter and if a pick-and-drop operation is unsuccessful, either picking or dropping, this counter is incremented. When the counter reaches $sleepMax$, then the cluster is considered to be sorted and a Boolean counter *sleeping* is true. The counter is reset to zero after a successful operation or if a new micro-cluster is placed in the cluster by a foreign Sorting Ant. When all ants are sleeping, the stop condition is met.

This step purifies each cluster and causes many smaller, similar clusters to dissolve and form one larger cluster. Clusters containing only one micro-cluster are considered to be outliers and the clustering solution is given as the set of non-empty clusters. Each cluster contains a grouping of density-reachable micro-clusters which summarize the partitioned areas of high-density in the feature space. These summary statistics are stored offline and the next tumbling window in the data stream is evaluated. This step is outlined in Algorithm 3.

IV. Experimental Study

The performance of ACSC is evaluated on both stationary and non-stationary datasets across three metrics. Since there are so few ant-based stream-clustering algorithms, ACSC is compared with four popular *static* ant clustering algorithms. ACA [42] extends the original pick-and-drop implementation [30] by introducing a cooling scheme for the picking probabilities. ACAm [6] extends this by associating a short term memory with each ant. The heuristic is further improved in ATTA [18] by using a colony of heterogeneous ants. *AntClust* does not use the pick-and-drop model for clustering but is instead inspired by the chemical recognition system of ants. The performance of these algorithms is taken from results already published in the literature. ACSC is then compared with three leading stream clustering algorithms on non-stationary streams. DenStream [7], CluStream [1] and ClusTree [26] are density based stream clustering algorithms. Similar to ACSC, they each employ micro-clusters to identify dense areas of the stream. However, unlike ACSC, they use the two-phase process of on-line summarisation and off-line clustering. *DenStream* uses a time dampened window to assign higher importance to more recent data and the off-line clustering phase is based on a variation of the DBSCAN [12] algorithm whereas *CluStream* applies a weighted *K-Means* algorithm on the generated micro-clusters. ClusTree uses a self-adaptive index structure to update the micro-clusters. Each of these algorithms are evaluated using the Massive Online Analysis (MOA) [5] open source software.

A. Performance Metrics

ACSC is evaluated across three metrics: Purity, F-Measure [22] and Rand Index [34]. Each dataset used is labelled and the ideal "correct" clustering solution is known, so performance is measured with respect to this ground truth. In each metric, a bad clustering will have a value close to 0 and an ideal clustering solution will have a value of 1.

The Purity metric measures how homogeneous a cluster is. A cluster is assigned to the class which appears most frequently within the cluster, the accuracy of this is evaluated by summing the instances of this class and dividing by the total number of instances in the cluster. The *F-Measure* (sometimes called *F-Score* or *F1-Score*) is the harmonic mean of the precision and recall scores obtained by the algorithm.

In the following, R represents the clustering result returned by the algorithm. R contains n clusters. In every identified cluster R_i ($i = \{1, \cdots, n\}$), V^i represents the most frequently appearing class label in cluster R_i, V^i_{sum} is the number of instances of V^i in R_i, and V^i_{total} represents the total number of instances of V^i in the current window. From these, we define the following features for cluster R_i:

$$precision_{R_i} = \frac{V^i_{sum}}{|R_i|} \tag{9}$$

$$recall_{R_i} = \frac{V^i_{sum}}{V^i_{total}} \tag{10}$$

$$Score_{R_i} = 2 * \frac{precision_{R_i} * recall_{R_i}}{precision_{R_i} + recall_{R_i}} \tag{11}$$

TABLE I: Description of datasets used in experiments

		Classes	Features	Examples	Drift Interval	Type
Non-Stationary						
	1CDT	2	2	16,000	400	Synthetic
	2CHT	2	2	16,000	400	Synthetic
	4CR	4	2	144,400	400	Synthetic
	4CE1CF	5	2	173,000	750	Synthetic
	Network Intrusion	2	42	494,000	unknown	Real
	Forest Cover	7	54	580,000	unknown	Real
Stationary						
	Iris	3	4	150	none	Real
	Wine	3	13	178	none	Real
	Zoo	7	17	101	none	Real

Overall, Purity (P) and F-Measure (F) can now be expressed in terms of the total number of clusters identified, as follows:

$$P = \frac{1}{n}\sum_{i=1}^{n} precision_{R_i} \tag{12}$$

$$F = \frac{1}{n}\sum_{i=1}^{n} Score_{R_i} \tag{13}$$

The Rand Index (R) is a measure of agreement between two clustering solutions; the solution identified by the algorithm and the *ideal* clustering solution known from the ground truth. It measures the number of decisions that are correct by penalising false negatives and false positives. Simply put, it measures the accuracy of the algorithm, defined as follows:

$$R = \frac{TP + TN}{TP + FP + TN + FN}, \tag{14}$$

where TP, TN, FP, and FP denote the number of true positive, true negative, false positive and false negative decisions, respectively.

We compare the performance of ACSC (stochastic) with results already published in the literature and also with three *deterministic* streaming algorithms (DenStream, CluStream and ClusTree). To statistically analyse the results on the above metrics, we use the non-parametric One-Sample Wilcoxon Signed-Rank Test [44]. We reject the null hypothesis that the distribution of the ACSC results are symmetric around the corresponding peer result with $p < 0.05$.

B. Datasets

ACSC is compared with other ant-based clustering solutions across three well known and popular non-stationary datasets; Iris, Wine, and Zoo. These datasets were taken from the UCI Machine Learning Depository[1] and the details of each are presented in Table I. These datasets are originally sorted by class so we randomly shuffle each dataset to remove any potential bias in the sorted streaming order. To evaluate the performance of ACSC over non-stationary streams, six datasets were used. Four datasets are synthetic and are taken from the non-stationary dataset used in [39] and made publicly available by the authors[2]. The four selected datasets were chosen in order to test increasing numbers of natural clusters present in the data. ACSC is also tested on two real data-streams: the Network Intrusion benchmark dataset[3] used in [15] and [7] and the Forest Cover-Type data-set[4]. The Network Intrusion data-stream is composed of seven weeks of simulated network requests on the DARPA network. Requests can be "normal" or "malicious". The non-stationary "malicious" class contains substantial drift as it is composed of twenty three different types of attack. The Forest Cover data-stream is composed of 54 cartographic variables describing forest coverage in Roosevelt National Forest of northern Colorado and is widely used in the stream-mining literature [1], [26], [16]. The full details of each dataset are presented in Table I. These datasets have been transformed into a stream by taking the input order as the streaming order. For the UCI datasets, we take the shuffled order as the streaming order.

TABLE II: Results of algorithms on stationary datasets regarding purity (P), F-Measure (F) and Rand Index (R)

Data		P	F	R
Iris				
	ACSC	$\mathbf{0.92}(\mathbf{s+})$	$\mathbf{0.90}(\mathbf{s+})$	$\mathbf{0.89}(\mathbf{s+})$
	ATTA	–	0.81	–
	ACA	0.77	0.77	0.78
	ACAm	0.81	0.81	0.81
	AntClust	0.89	0.84	0.84
Wine				
	ACSC	$\mathbf{0.94}(\mathbf{s+})$	$\mathbf{0.90}(\mathbf{s+})$	$\mathbf{0.88}(\mathbf{s+})$
	ATTA	–	0.88	–
	ACA	0.86	0.86	0.83
	ACAm	0.86	0.87	0.85
	AntClust	0.94	0.73	0.73
Zoo				
	ACSC	$\mathbf{0.97}(\mathbf{s+})$	$\mathbf{0.85}(\mathbf{s+})$	0.88(s-)
	ATTA	–	0.81	–
	ACA	0.77	0.76	0.88
	ACAm	0.76	0.77	0.89
	AntClust	0.66	0.68	**0.90**
Average				
	ACSC	$\mathbf{0.94}(\mathbf{s+})$	$\mathbf{0.90}(\mathbf{s+})$	$\mathbf{0.88}(\mathbf{s+})$
	ATTA	–	0.83	–
	ACA	0.80	0.80	0.83
	ACAm	0.81	0.82	0.85
	AntClust	0.83	0.75	0.82

C. Clustering Quality Evaluation

To evaluate ACSC on the static datasets the algorithm was tested on one window with $windowSize$ set to the number of samples. ACSC is compared with leading ant clustering

[1] http://archive.ics.uci.edu/ml/

[2] https://sites.google.com/site/nonstationaryarchive/

[3] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[4] https://archive.ics.uci.edu/ml/datasets/covertype

TABLE III: Average performance of algorithms over the entire stream regarding purity (P), F-Measure (F) and Rand Index (R)

	DenStream			*CluStream*			*ClusTree*			*ACSC*		
	P	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*
$1CDT$	0.99	0.82	0.77	**1.0**	0.88	0.80	**1.0**	0.89	0.82	0.99(s-)	**0.99**(s+)	**0.99**(s+)
$2CHT$	0.43	0.27	0.53	0.24	0.23	0.55	0.22	0.24	**0.58**	**0.81**(s+)	**0.42**(s+)	0.55(s-)
$4CR$	**1.00**	0.67	0.71	**1.00**	0.89	0.89	**1.00**	0.89	0.89	0.99(s-)	**0.95**(s+)	**0.97**(s+)
$4CE1CF$	**0.99**	0.35	0.56	**0.99**	0.86	0.89	**0.99**	0.86	0.89	0.96(s-)	0.76(s-)	**0.90**(s+)
$Network$	**1.00**	0.80	0.81	0.35	0.13	0.36	0.36	0.16	0.3	**1.0**(=)	**0.95**(s+)	**0.95**(s+)
$CoverType$	**0.89**	0.10	0.51	0	0	0	0	0	0	0.88(s-)	**0.59**(s+)	**0.64**(s+)
Average	0.88	0.50	0.64	0.59	0.49	0.58	0.59	0.51	0.58	**0.93**	**0.77**	**0.83**

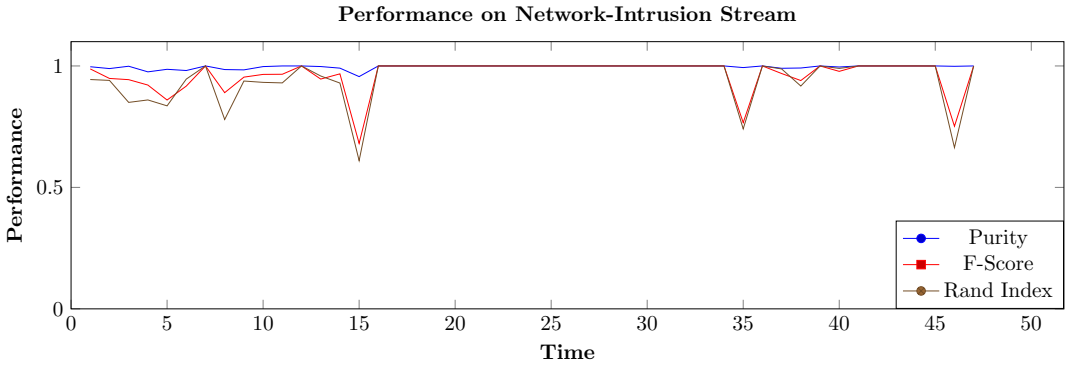

Fig. 1: Performance of ACSC on 4CR dataset over 100 windows of size 1,000, where $\epsilon = 0.05$

algorithms and the performances of these algorithms were taken from results already published in the literature, with the exception of our implementation of AntClust. The Purity and Rand Index for the ATTA algorithm are omitted as they are not available in the literature. The comparative results are presented in Table II. The average of 30 runs of ACSC is presented along with result of the Wilcoxon Signed-Rank Test, where "$s+$" indicates ACSC performs significantly better than the best peer result and "$s-$" indicates ACSC performs significantly worse. ACSC performs significantly better on each metric in the Iris and Wine datasets and is outperformed only by *AntClust* on the Rand Index measure on the Wine dataset. The overall average shows that ACSC outperforms the others on these three datasets on all metrics.

To evaluate ACSC on non-stationary streams, ACSC is compared with DenStream, CluStream and CluStream. Table III displays a comparative evaluation of each algorithm across the entire stream. The peer algorithms are deterministic but ACSC is stochastic so the displayed results are the average, along with the Wilcoxon test over 30 runs. ACSC, on average over all 6 datasets, outperforms the compared algorithms. The levels of cluster purity are comparable across each dataset except for 2CHT where ACSC performs much better. ACSC achieves the best Rand Index scores on each dataset and the best F-Measure on five out of six datasets and on average, is the best overall. On the final stream; Forest Cover, we are using the full dataset, containing 54 variables, both continuous and discreet. CluStream and ClusTree were unable to find a clustering solution on this full dataset (we are evaluating through the MOA platform). Previous studies report using a subset of the data (the first 10 continuous variables).

While Table III shows the mean values across the whole stream, the on-line performance of ACSC as a stream progresses is displayed in Figs. 1 and 2. We use the two real data-streams (Network Intrusion and Forest Cover) and plot the algorithm's performance over time.

To evaluate the time requirement of the algorithm we measure the speed of the algorithm in seconds. We measure the total time the algorithm takes to process a stream. We also report the average time to process a single window with size 1,000. Through MOA, we also measure the performance of the peer algorithms for comparison and report the results in Table IV.

Fig. 3 shows the mean number of calculations performed by ACSC and, for comparison, DenStream on the Network Intrusion stream. These comparisons are the Euclidean distance comparisons between two micro-clusters. Intuitively, the greater the number of comparisons, the longer the algorithm takes. The sampling method in ACSC, and the stage at which micro-clusters attempt to merge mean that far fewer comparisons are needed and the reason why ACSC can process a stream comparatively faster. These results are based on a window size of 1,000 for ACSC and a horizon and $initPoints$ of 1,000 for DenStream. The other DenStream variables were tuned to $\epsilon = 0.02$, $\beta = 0.2$, $\mu = 1$, and $\lambda = 0.25$.

The memory requirement of the algorithm is a function of the window size. The window is read in a single pass, summarized into a smaller number of micro-clusters and then deleted. These micro-clusters are operated on and stored. So the overall memory usage is determined by the size of the window (the

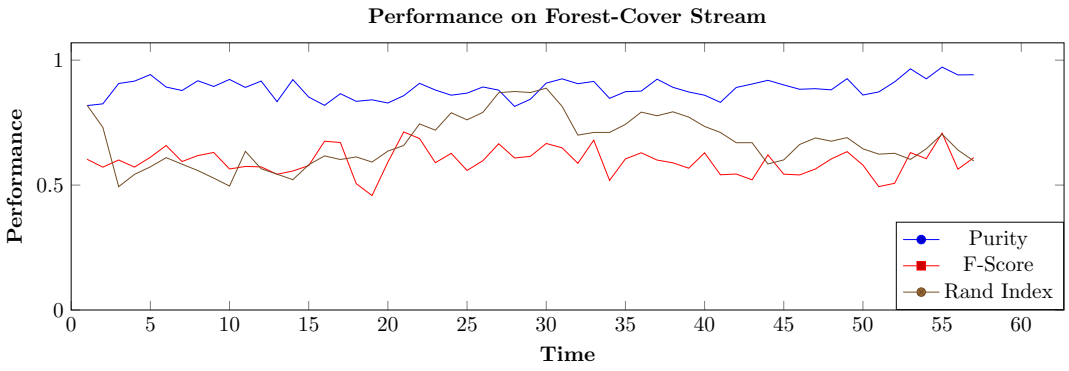

Fig. 2: Performance of ACSC on 4CR dataset over 100 windows of size 1,000, where $\epsilon = 0.05$

TABLE IV: Time measurement in seconds over stream. Total time for stream and average window length using window size of 1,000. * Faster algorithms did not discover a clustering solution

	DenStream		*CluStream*		*ClusTree*		*ACSC*	
	Total,	Window	Total,	Window	Total,	Window	Total,	Window
$1CDT$	05.74	0.38(0.06)	01.69	0.11(0.02)	01.22	0.07(0.01)	**0.71**(0.01)	**0.05**(0.02)
$2CHT$	05.61	0.37(0.05)	01.67	0.11 (0.02)	01.38	0.09 (0.02)	**0.62** (0.06)	**0.05**(0.02)
$4CR$	50.62	0.29(0.04)	11.78	0.09(0.01)	12.11	0.09(0.01)	**09.28**(0.1)	**0.06**(0.01)
$4CE1CF$	55.06	0.38(0.03)	14.64	0..08(0.01)	**12.96**	**0.08**(0.41)	16.85(0.3)	0.09(0.01)
$Network$	94.41	0.19(0.77)	106.21	0.22(0.18)	22.11	0.06(0.3)	**20.63**(0.3)	**0.04**(0.02)
$CoverType$	278.5	0.56(0.09)	26.62	0.04(0.02)*	22.07	0.03(0.02)*	**49.53**(1.07)	**0.08**(0.02)

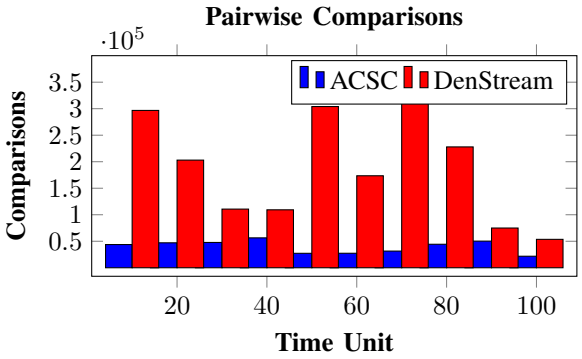

Fig. 3: Mean number of calculations performed on Network Intrusion dataset over 100 windows of size 1,000.

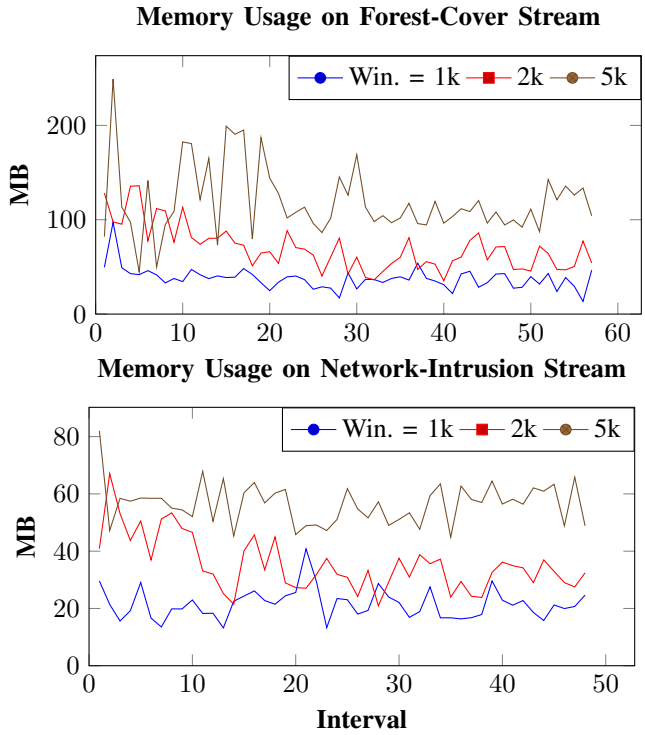

Fig. 4: Memory usage in MB.

number and dimensionality of points) loaded into memory. We use a commercial profiler [21] to accurately measure the memory usage of the algorithm as a stream progresses. We report the usage in MB on the Network Intrusion and Forest Cover data streams. We measure memory usage on different window sizes of 1k, 2k and 5k. These results are displayed in Fig. 4. For display purposes, each plot-point is an average over a set of windows. For windows of length 1k, we averaged 10 windows. For windows of length 2k, we averaged 5 windows, and for windows of length 5k, we averaged 2 windows. It can be seen that as the window size increases, the memory usage increases. Also, as the dimensionality increases (Forest Cover), the memory usage increases. We used a PC with an Intel processor at 2.6GHz and 8GB of RAM.

TABLE V: Effect of the nComp parameter on speed and performance, Network Intrusion stream with varying window sizes

Ncomp.	$Win. = 1,000$		$Win = 2,000$		$Win = 5,000$	
	$Perform.$	$RunTime$	$Perform.$	$RunTime$	$Perform.$	$RunTime$
0.05	0.955 (0.2)	15.3 (0.517)	0.951 (0.81)	35.5 (13.5)	0.943 (1.8)	115.4 (4.30)
0.1	0.961 (0.02)	17.9 (0.565)	0.957 (0.02)	36.02 (0.42)	0.946 (0.06)	121.35 (8.20)
0.2	0.965 (0.01)	22.2 (0.168)	0.957 (0.01)	45.07 (0.83)	0.946 (0.01)	125.5 (2.79)
0.4	0.967 (0)	30.2 (0.183)	0.958 (0.01)	59.3 (0.19)	0.946 (0)	145.61 (5.71)
0.8	0.968 (0)	32.9 (0.140)	0.958 (0)	71.8 (1.2)	0.947 (0)	217.01 (6.1)
1.0	0.968 (0)	38.8 (0.031)	0.958 (0)	76.0 (0.7)	0.947 (0)	247.31 (1.19)

TABLE VI: Comparison of DenStream with stochastic and deterministic ACSC. Network-Intrusion, window = 1,000

	Time (secs.)	Purity	F1	Rand Index
$DenStream$	94.41	1.0	0.80	0.81
$ACSC_{deter.}$	38.8	1.0	0.96	0.95
$ACSC_{stoc.}$	17.9	1.0	0.95	0.95

TABLE VII: Comparison of DenStream with stochastic and deterministic ACSC. Forest-Cover, window = 1,000

	Time (secs.)	Purity	F1	Rand Index
$DenStream$	278.5	0.89	0.10	0.51
$ACSC_{deter.}$	71.83	0.88	0.59	0.64
$ACSC_{stoc.}$	49.53	0.88	0.59	0.64

TABLE VIII: Wine clustering solution before Sorting Ants

Nest	class 1	class 2	class 3
1	[59	9	0]
2	[0	51	6]
3	[0	8	0]
4	[0	1	0]
5	[0	2	0]
6	[0	0	40]
7	[0	0	2]

TABLE IX: Wine clustering solution after Sorting Ants

Nest	class 1	class 2	class 3
1	[59	9	0]
2	[0	62	6]
3	[0	0	42]

D. Effect of Stochastic Sampling and Sample-Size

In phase one of the algorithm we make a single pass of a window, incrementally forming clusters. We evaluate a point p's similarity with an existing cluster C using the Euclidean distance from p with a sample (without replacement) from C. The $nComp$ parameter determines the size of this sample:

$$nSamples = WindowSize * nComp. \tag{15}$$

A smaller value for $nComp$, say 0.05, will result in a smaller sample taken, fewer comparisons made and, intuitively, a faster run. A larger value for $nComp$ will require more comparisons, slowing the algorithm but offering, potentially, greater accuracy with less variance in results. A value of 1.0 for $nComp$ requires a comparison with each point in every cluster (not just a sample) effectively making this phase of the algorithm deterministic.

In Table V we report the performance (over 10 runs) of ACSC on the Network Intrusion data-stream with gradually increasing values for $nComp$. We report the performance of the algorithm (the average of the Purity, F1-Measure and Rand Index metrics) along with the total running time of the algorithm (in seconds). We report the results on this stream with varying window sizes.

On this data-stream it can be seen that the performance of the algorithm improves only very slightly with an increase in $nComp$ whereas the running time increases with larger values. For example, with each window size, a value of 1.0 takes over twice as long as a value of 0.1 with only a minimal improvement in performance.

In Tables VI and VII we compare the results of DenStream with ACSC on two real data-streams; Network Intrusion and Forest-Cover, respectively. We report ACSC using a deterministic phase ($nComp = 1.0$) and, alternatively, a stochastic phase ($nComp = 0.1$). The deterministic ACSC performs better than DenStream on the F1 and Rand Index metrics and is also faster. The stochastic version is faster still, with an almost identical performance.

E. Performance of Sorting Ants

In this section, we examine the effect of the second phase of ACSC. In the first phase, rough clusters are formed. In the second phase, Sorting Ants are assigned to each cluster and inter-cluster sorting is performed. We examine the effect of these Sorting Ants on the initial clusters.

The stopping condition for this phase is determined by the $sleepMax$ parameter; the number of unsuccessful sorting attempts allowed for each ant before it is "asleep". A $sleepMax$ of 0 means that this phase is never performed. We use the Wine dataset and a window from the Network Intrusion stream as illustrative examples of the effect of the Sorting Ants. The Wine dataset contains three classes with a distribution of [59, 71, 48]. While the window (size = 1,000) from the Network Intrusion stream has a distribution of [998, 2]. Table VIII shows the clusters identified in phase one on the Wine dataset. The three natural clusters are grouped into 7 clusters with an overall purity = 0.97, F-Score = 0.86 and Rand Index = 0.85.

Table IX shows the clusters after the Sorting phase with a $sleepMax$ of 3. Three clusters are identified with an overall purity = 0.94, F-Score = 0.91 and Rand Index = 0.90.

TABLE X: Network Intrusion window before Sorting Ants

Nest	class 1	class 2
1	[841	0]
2	[48	0]
3	[31	2]
3	[78	0]

TABLE XI: Network Intrusion window after Sorting Ants

Nest	class 1	class 2
1	[857	0]
2	[70	0]
3	[1	2]
3	[70	0]

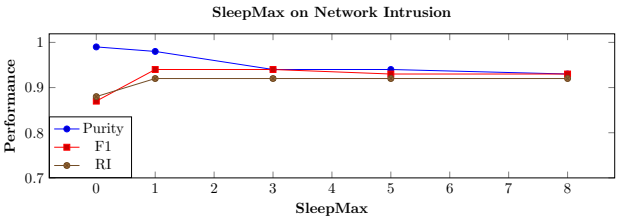

Fig. 5: Sleep Max on Network Intrusion.

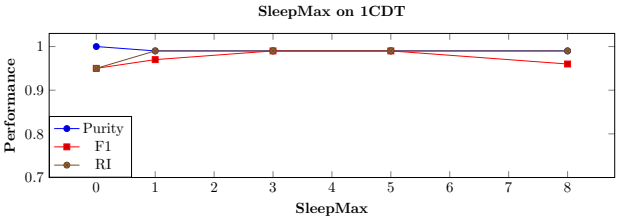

Fig. 6: Sleep Max on 1CDT.

Table X shows the Network Intrusion window before, with purity = 0.98, F1 = 0.51, and Rand Index = 0.72. The sorted clusters are displayed in Table XI. The sorted clusters have a similar purity but higher F1 and Rand Index score (0.86 and 0.75, respectively).

Imitating their biological counterparts, the Sorting Ants are biased to picking-up isolated items and dropping them in denser areas. The final clusters are a closer representation of the true underlying structure. The purity score is lower than the initial clusters as the *average* purity is measured. For example, cluster 3 in Table VIII contains a single micro cluster and so has 100% purity and the overall average increases. However, these sparse clusters lower the F-Score and Rand Index metrics. Taken on its own, Purity can be a misleading metric as it does not consider the actual topology of the data. A similar performance can be seen in the Network Intrusion dataset (first 100 windows, with a window size of 1,000) and the 2CDT dataset in Figs. 5 and 6, respectively.

In previous studies [6], [18], [19], [30], the decisions to pick and drop are probabilistic. We evaluate a deterministic alternative to these probabilistic operations. Because the initial clusters in the first part of the algorithm are formed using a stochastic process, the algorithm itself is non-deterministic and

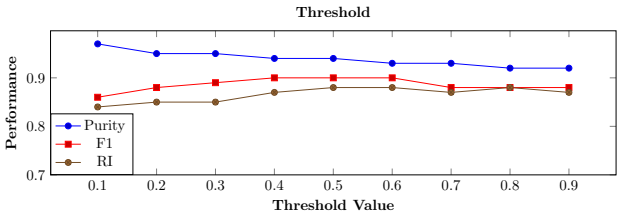

Fig. 7: threshold values on Wine

TABLE XII: Sorting Ants comparison using purity (P), F-Measure (F) and Rand Index (R)

Data Stream	*Prob.*			*Deter.*		
(threshold)	P	F	R	P	F	R
1CDT (0.4)	0.99	0.99	0.99	0.99	0.99	0.99
Network (0.4)	0.99	0.95	0.94	0.99	0.96	0.94
Wine (0.5)	0.94	0.90	0.88	0.94	0.89	0.89

the comparisons here relate only to the behavior of the Sorting Ants. The decision for picking a micro-cluster (see Eq. (8)) is a probability based on the number of density-reachable micro-clusters in the cluster and the parameter $nComp$. We introduce a deterministic threshold to replace the probability. The deterministic decision to pick a micro-cluster we propose is given as follows:

$$\frac{reachable}{nComp} \geq nComp * threshold \tag{16}$$

A drop is successful if there are more reachable micro-clusters in the second cluster than there are in the first cluster. To find a suitable threshold, we evaluate different values on 30 runs of the Wine dataset and the results are presented in Fig. 7. The best performance using a deterministic decision with a threshold of 0.5 on the Wine dataset is very similar to the probabilistic decisions, a similar observation can be made on other datasets, as shown in Table XII. Overall, the results are very similar and we elect to use the probabilistic functions as they do not require an additional tunable parameter

F. Scalability and Robustness to Noise

To test the scalability of ACSC, we generated synthetic data sets with different numbers of dimensions and different numbers of natural clusters. As in [7], the points in each synthetic data set are drawn from a series of Gaussian distributions (each representing a cluster). The mean and variance of each distribution are changed after every 5,000 points during the data generation process. We follow the notation used in [7] to describe the synthetic data sets: 'B' indicates the number of data points (in hundreds of thousands), 'C' and 'D' indicate the number of clusters present and the dimensionality of each point, respectively. For example, B2C10D20 indicates the data set contains 200,000 data points of 20 dimensions, belonging to 10 different clusters.

The performance of the algorithm is measured in the execution time using a window size of 10,000 and $\epsilon = 0.05$. The scalability of the algorithm, in terms of time, is evaluated

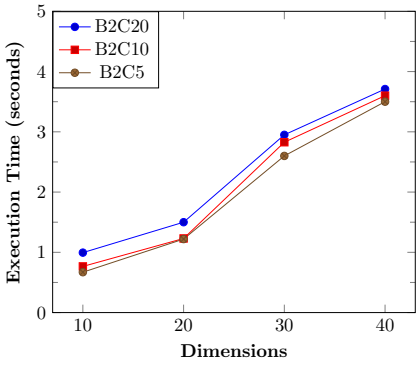

Fig. 8: Scalability to the number of dimensions.

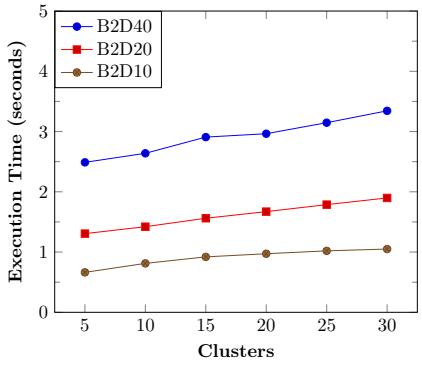

Fig. 9: Scalability to the number of clusters.

against increasing number of clusters and also increasing number of dimensions. First, the number of data points and clusters are fixed and the algorithm is tested on varying numbers of dimensions from 10 to 40. In Fig. 8, it can be seen that as the dimensionality increases, the execution time increases linearly. The plot follows a similar trend for each dataset regardless of how many clusters are present suggesting that the dimensionality is a more important factor than the number of clusters. This is confirmed in Fig. 9. The number of data-points and dimensions are fixed and the number of clusters increases from 5 to 30. As the amount of clusters increases, the execution time increases only marginally.

To evaluate how robust the algorithm is to noise, we introduce random samples to three datasets; B1C5D20, Wine, and Network Intrusion (first 100 windows of size 1,000). To introduce 5% noise, we replace 5% of the final dataset with random samples. Tables XIII, XIV, and XV show the average performance over 30 runs of ACSC on each dataset with varying levels of noise.

The results show that it is robust and the performance of the algorithm is not greatly affected by noise. It is interesting to note that the number of clusters identified by the algorithm increases with the number of noisy samples. Each random point is assigned to its own cluster and the natural clusters remain relatively unaffected.

TABLE XIII: B1C5D20 with noise, $\epsilon = 0.05$

Noise	Purity	F-Measure	R. Index	#Nests
0%	1.0	1.0	1.0	5.0
3%	1.0	1.0	1.0	35.7
5%	1.0	1.0	1.0	54.1
8%	0.99	0.98	0.98	82.4

TABLE XIV: Wine with noise, $\epsilon = 0.07$

Noise	Purity	F-Measure	R. Index	#Nests
0%	0.94	0.9	0.88	4.1
3%	0.94	0.9	0.88	8.8
5%	0.93	.89	0.86	15.4
8%	0.91	0.85	0.87	20.1

TABLE XV: Network Intrusion with noise, $\epsilon = 0.09$

Noise	Purity	F-Measure	R. Index	#Nests
0%	0.99	0.95	0.94	1.6
3%	0.99	0.95	0.94	30.8
5%	0.99	0.93	0.94	52.0
8%	0.98	0.91	0.93	79.2

G. Sensitivity Analysis

In previous sections weve examined the sensitivity of the $nComp$ parameter (Section IV-D) and the $sleepMax$ parameter (Section IV-E). In this section we examine the sensitivity of the ϵ parameter and the effect of different window sizes on the algorithm's performance. The ϵ-neighbourhood is crucial in density clustering: if it is too small, no clusters will form; if it is too large, there will be rough and impure clusters. This value is sensitive and data-dependent. Figs. 10 and 11 show the sensitivity of the parameter on the Network intrusion and 4CE1CF datasets, respectively.

To evaluate the sensitivity of window sizes, ACSC was tested across 6 different window sizes: 500, 1000, 1500, 2000, 3000 and 5000. The mean Purity, F-Measure and Rand Index are calculated across each window size in the stream and, for visualisation purposes, we report the algorithm's 'score', which is simply the average of all three metrics. The results are shown in Fig. 12. On both Network Intrusion and 4CE1CF, it can be seen that the window size has the minimal effect on the accuracy of the algorithm.

H. Discussion

ACSC is shown to outperform other ant-based clustering algorithms over three static datasets. It also, on average, outperforms DenStream, CluStream and ClusTree over all 6 non-stationary datasets. The levels of cluster purity are comparable across each dataset but purity, in isolation, is not a very revealing evaluation metric as it does not consider the true topology of the data, for example, assigning each data point to its own cluster would give 100% purity. It is, however, a useful metric when taken alongside the F-Measure and Rand Index measures. ACSC achieves the best F-Measure and Rand Index scores on five out of six datasets and, on average, performs the best overall. The second phase of the algorithm, the sorting phase, is the reason for this. The

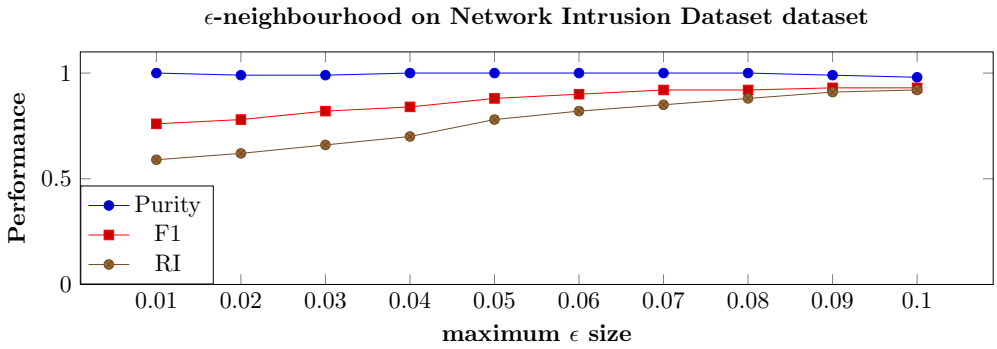

Fig. 10: Sensitivity of ϵ-neighbourhood on Network Intrusion Stream.

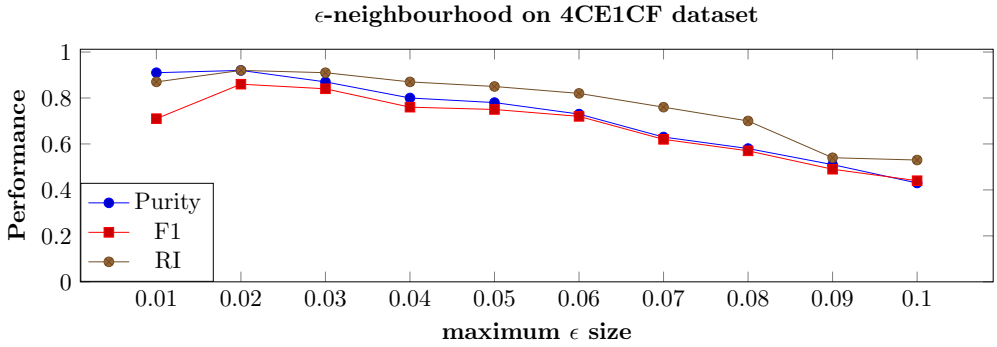

Fig. 11: Sensitivity of ϵ-neighbourhood on 4CE1CF Stream.

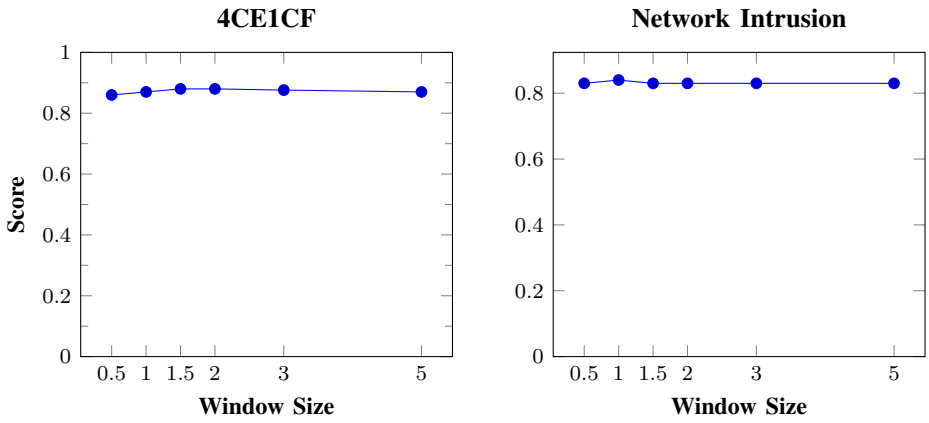

Fig. 12: Sensitivity of window size (in thousands).

probabilistic functions for picking and dropping micro-clusters are biased towards the dissolution of smaller clusters and incorporating them into similar, larger clusters. This improves the precision and recall scores (and hence the F-Measure) and creates clusters closer to the "true" structure of the data. This is reflected in the Rand Index score. ACSC can process the evaluated streams faster than the comparative peer algorithms. This is due to the stochastic sampling method and also how micro-clusters attempt to merge. This merging operation is expensive. In ACSC, only micro-clusters in the same cluster attempt to merge, replacing an exhaustive search and reducing the number of failed merging operations. Of the real data-streams evaluated, DenStream performs better than CluStream and ClusTree. However, ACSC outperforms DenStream.

It is interesting to note that ACSC performs favourably but requires fewer parameters and considerably fewer calculations. Fig. 3 shows that ACSC requires roughly 10 times fewer calculations. This is due to the fact that DenStream performs an exhaustive search for the nearest neighbour of each micro-cluster. If ACSC used a deterministic implementation whereby each point is compared with every other point, it would require O(N^2) time. But, each point in ACSC is evaluated against a sample taken from a cluster. The absolute worst case would require O(N^2) only if n data points in each window belonged to n different clusters. Experimental results show that the number of calculations is comparatively low. We report that for the Network Intrusion stream, with 42 dimensions, the algorithm can process a window of 1,000 points in, on average,

0.04 seconds with an average memory requirement of 21.5 MB. The larger Forest-Cover stream can be processed in, on average, 0.08 seconds while requiring 37.6 MB of memory.

ACSC uses a stochastic sampling-without-replacement method in the initial phase and a probabilistic pick-and-drop model for the Sorting Ants in the second phase. However, it is possible to run ACSC deterministically. The sampling rate in the first phase is determined by the parameter $nComp$. We investigate the effects of varying this parameter in IV-D. A lower value for this parameter (we use 0.1 in all experiments) reduces the run-time with minimal effect on performance. If the sampling rate is set to $WindowSize$, then the first phase is deterministic, but it will be slower without any further improvement in the clustering accuracy. In the second phase, the probabilistic Sorting Ants can be replaced with deterministic Sorting Ants described in Section IV-E. With these changes, the algorithm is deterministic but its speed is sacrificed and an additional, data-dependant parameter $threshold$ is required.

The window size has little effect on the accuracy of the algorithm. This window size is just one of three tune-able parameters required (unlike DenStream, for example, which requires six parameters). The second parameter is the sleepMax value, which determines how many unsuccessful sorting attempts are allowed before a cluster is considered to be sorted. Experimental results show that the performance plateaus after a $sleepMax$ limit of three. Any value greater than this gives similar results but requires additional, unnecessary, computation as each ant continues to attempt sorting the cluster. The final parameter is the ϵ-neighbourhood. This is data dependent and very sensitive as shown in Figs. 10 and 11.

V. Conclusions

In this paper we proposed an Ant Colony Stream Clustering (ACSC) algorithm for clustering dynamic data streams. ACSC uses the tumbling window model and results show that it scales linearly to larger window sizes and higher dimensionality, while being robust to noise. Clusters are formed in a single pass of the data using a stochastic sampling method. The sampling method replaces an exhaustive search and is shown to require considerably fewer calculations. The deterministic method (corresponding to *nComp*=1.0) yields the highest performance, at the cost of the longest run time. With a suitable choice of the parameter *nComp*, the proposed algorithm achieves a significant speed up at only little performance loss. The initial clusters discovered are further refined using a method inspired by the sorting behaviour of ants. This sorting method is based on the classic pick-and-drop ant clustering algorithm. The probabilistic functions for picking and dropping are biased towards the dissolution of smaller clusters and incorporating their contents into similar, larger clusters. This improves the precision and recall scores and creates clusters closer to the "true" structure of the data. Our implementation addresses a short-coming of the original pick-and-drop model; speed. Rough clusters are identified quickly in a single pass and *then* sorted. Furthermore, in the traditional algorithm, data points are moved individually which can take a long time. By grouping similar points into micro-clusters, a number of points can be moved in a single operation, further speeding the algorithm.

ACSC was shown to outperform other ant-based clustering algorithms in the literature and was compared with three popular stream clustering algorithms across real and synthetic datasets. Experimental results show that ACSC performs favourably while requiring fewer parameters. Of the required parameters, the ϵ parameter was shown to be very sensitive and greatly affects the performance of ACSC. It is data-dependent and requires manual tuning. Furthermore, it is global and so restricts the algorithm to finding clusters of similar density, a common problem for density based clustering algorithms. Further research will investigate an adaptive, local ϵ parameter. This could potentially allow the discovery of clusters with varying densities in the data.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," Proc. 29th Int. Conf. Very Large Data Bases, vol. 29, pp. 81–92, 2003.

[2] H. Azzag, "AntTree: a new model for clustering with artificial ants," Proc. 2003 IEEE Conf. Evol. Comput., vol. 4, 2003.

[3] R. D. Baruah and P. Angelov, "DEC: Dynamically evolving clustering and its application to structure identification of evolving fuzzy models," IEEE Trans. Cybern., vol. 44, no. 9, pp. 1619–1631, Sep. 2014.

[4] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM: The fuzzy c-means clustering algorithm," Computers and Geosciences, vol. 10, no. 2–3, pp. 191–203, Jan. 1984.

[5] A. Bifet, G. Holmes, R. Kirby, and B. Pfahringer, "MOA: Massive online analysis," Journal of Machine Learning Research, vol. 11, pp. 1601–1604, 2010.

[6] U. Boryczka, "Finding groups in data: Cluster analysis with ants," Applied Soft Computing, vol. 9, no. 1, pp. 61–70, Jan. 2009.

[7] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," SDM, vol. 6, pp. 328–339, 2006.

[8] Y. Chen, & L. Tu, "Density-based clustering for real-time stream data." In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 133-142). ACM. August, 2007

[9] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien, "The dynamics of collective sorting robot-like ants and ant-like robots," Proceedings of the 1st International Conference on Simulation of Adaptive Behavior From Animals to Animats, pp. 356–363, 1991.

[10] D. Deng and N. Kasabov, "ESOM: An algorithm to evolve self-organizing maps from online data streams". Neural Networks, Proceedings of the IEEE-INNS-ENNS International Joint Conference on (Vol. 6, pp. 3-8). IEEE. 2000

[11] M. Dorigo, M. Birattari, and T. Stizle,"Ant colony optimization," IEEE Comput. Intell. Mag., vol. 1, no. 4, pp. 28–39, 2006.

[12] M. Ester, H. P. Kriegel, J. Sander, and X. Xu,"A density-based algorithm for discovering clusters in large spatial databases with noise," KDD, vol. 96, pp. 226–231, 1996.

[13] A. H. Fahim, A. M. Salem, F. A. Torkey, and M. A. Ramadan, "Density Clustering Based on Radius of Data (DCBRD)," World Academy of Science, Engineering and Technology, 2006.

[14] C. M. Fernandes, A. M. Mora, J. J. Merelo, and A. C. Rosa, "KANTS: A Stigmergic ant algorithm for cluster analysis and swarm art," IEEE Trans. Cybern., vol. 44, no. 6, pp. 843–856, Jun. 2014.

[15] A. Forestiero, C. Pizzuti, and G. Spezzano, "A single pass algorithm for clustering evolving data streams based on swarm intelligence," Data Mining and Knowledge Discovery, vol. 26, no. 1, pp. 1–26, Nov. 2011.

[16] M. Ghesmoune, M. Lebbah and H. Azzag, " A new growing neural gas for clustering data streams" Neural Networks, 78, pp.36-50. 2016.

[17] S. Guha, and N. Mishra. "Clustering data streams." Data Stream Management. Springer Berlin Heidelberg, pp. 169-187, 2016.

[18] J. Handl, J. Knowles, and M. Dorigo, "Ant-based clustering and topographic mapping," Artif. Life, vol. 12, no. 1, pp. 35–62, Jan. 2006.

[19] J. Handl and B. Meyer, "Ant-based and swarm-based clustering," Swarm Intell., vol. 1, no. 2, pp. 95–113, Nov. 2007.

[20] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," Applied Statistics, vol. 28, no. 1, pp. 100, 1979.
[21] J-Profiler: Java Profiler. https://www.ej-technologies.com/products/jprofiler/overview.html, 21 11 2017.
[22] N. Jardine and C. J. van Rijsbergen, "The use of hierarchic clustering in information retrieval," Information Storage and Retrieval, vol. 7, no. 5, pp. 217–240, Dec. 1971.
[23] H. Jiang, J. Li, S. Yi, X. Wang, and X. Hu, "A new hybrid method based on partitioning-based DBSCAN and ant clustering," Expert Syst. with Appl., vol. 38, no. 8, pp. 9373–9381, Aug. 2011.
[24] P. M. Kanade and L. O. Hall, "Fuzzy ants and clustering," IEEE Trans. Syst., Man, and Cybern. - Part A: Syst. and Humans, vol. 37, no. 5, pp. 758–769, Sep. 2007.
[25] M. Korrek and A. Nizam, "A new arrhythmia clustering technique based on ant colony optimization," J. of Biomedical Inform., vol. 41, no. 6, pp. 874–881, Dec. 2008.
[26] P. Kranen, I. Assent, C. Baldauf and T. Seidl "The ClusTree: indexing micro-clusters for anytime stream mining" Knowledge and information systems, 29(2), pp.249-272. 2011
[27] N. Labroche, N. Monmarche, and G. Venturini, "AntClust: ant clustering and web usage mining," Pro. 2003 Genetic and Evol. Comput. Conf., pp. 25–36, 2003.
[28] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. "Semantics and evaluation techniques for window aggregates in data streams." Proc. 2005 ACM SIGMOD Int. Conf. on Management of data, pp. 311-322, 2005.
[29] B. Liu, "A Fast Density-Based Clustering Algorithm For Large Databases," Proc. 5th Int. Conf. Machine Learning and Cybern., 2006.
[30] E. Lumar and B. Faieta, "Diversity and adaptation in populations of clustering ants," Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior: From Animals to Animats, vol. 3, pp. 489–508, 1994.
[31] S. Mahran and K. Mahar, "Using grid for accelerating density based clustering," Proc. 2008 IEEE Int. Conf. Computer and Inform. Tech., 2008.
[32] N. Masmoudi, H. Azzag, M. Lebbah, B. Cyrille, and B. J. Maher, "How to use ants for data stream clustering," Proc. 2015 IEEE Cong. Evol. Comput., pp. 656–663, 2015.
[33] P. Hore, L.O. Hall, and D. B. Goldgof. "Creating streaming iterative soft clustering algorithms." IEEE Fuzzy Information Processing Society, 2007. NAFIPS'07. Annual Meeting of the North American, 2007.
[34] W. M. Rand, "Objective criteria for the evaluation of clustering methods," J. of the American Statistical Assoc., vol. 66, no. 336, pp. 846, Dec. 1971.
[35] S. U. Rehman, A. Ashgar, S. Fong, and S. Sarasvady, "DBSCAN: Past, present and future," Proc. 5th Int. Conf. Appl. of Digital Information and Web Technologies (ICADIWT), pp. 232–238, 2014.
[36] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," ACM SIGGRAPH Computer Graphics, vol. 21, no. 4, pp. 25–34, Aug. 1987.
[37] T. A. Runkler, "Ant colony optimization of clustering models," Int. J. of Intell. Syst., vol. 20, no. 12, pp. 1233–1251, 2005.
[38] P. S. Shelokar, V. K. Jayaraman, and B. D. Kulkarni, "An ant colony approach for clustering," Analytica Chimica Acta, vol. 509, no. 2, pp. 187–195, May 2004.
[39] M. T. Chao, "Data stream classification guided by clustering on non-stationary environments and extreme verification latency," Proc. 2015 SIAM Int. Conf. Data Mining, pp. 873–881, 2015.
[40] P.-N. Tan, M. Steinbach, and V. Kumar, Introduction to Data Mining. Boston, MA: Pearson/Addison-Wesley, 200, ch. 9, pp. 147–160.
[41] L. Tu and Y. Chen, "Stream data clustering based on grid density and attraction," ACM Trans. Knowledge Discovery from Data, vol. 3, no. 3, pp. 1–27, Jul. 2009.
[42] A. L. Vizine, L. De Castro, and R. Gudwin, "Text document classification using swarm intelligence," Proc. of KIMAS, 2005.
[43] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang, "Density-based clustering of data streams at multiple resolutions," ACM Trans. Knowledge Discovery from Data, vol. 3, no. 3, pp. 1–28, Jul. 2009.
[44] F. Wilcoxon,and R. A. Wilcox. "Some rapid approximate statistical procedures". Lederle Laboratories, 1964.
[45] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: A new data clustering algorithm and its applications," Data Mining and Knowledge Discovery, vol. 1, no. 2, pp. 141–182, 1997.

Conor Fahy received a B.Sc. degree in Computer Science from Dublin City University, Ireland in 2004. He received an M.Sc. degree in Intelligent Systems from De Montfort University, Leicester, UK in 2016, where he is currently pursuing a Ph.D. degree in the Centre for Computational Intelligence. His research interests include swarm intelligence and ensemble methods for unsupervised and semi-supervised learning in dynamic environments.

Shengxiang Yang (M'00–SM'14) received the B.Sc. and M.Sc. degrees in automatic control and the Ph.D. degree in systems engineering from Northeastern University, Shenyang, China in 1993, 1996, and 1999, respectively. He is currently a Professor in Computational Intelligence and Director of the Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, Leicester, U.K. He has over 250 publications. His current research interests include evolutionary computation, swarm intelligence, artificial neural networks, data mining and data stream analysis, and relevant real-world applications. He serves as an Associate Editor/Editorial Board Member of seven international journals, such as the *IEEE Transactions on Cybernetics*, *Information Sciences*, *Evolutionary Computation*, and *Soft Computing*.

Mario Gongora received the Ph.D. degree from the University of Warwick, U.K. He is currently an Associate Professor with the School of Computer Science and Informatics, De Montfort University. He is also the Deputy Director of the Centre for Computational Intelligence. His research includes the application of artificial intelligence techniques to the identification, modeling, simulation, and control of complex systems. His expertise is mainly in using evolutionary computing and biologically inspired methods for this purpose. He is also involved in close contact with industry, applying his research results in the analysis of consumer behavior and other complex industrial processes.