Received <day> <Month>, <year>; | Revised <day> <Month>, <year>; | Accepted <day> <Month>, <year>

DOI: xxx/xxxx

RESEARCH ARTICLE

Data Stream Classification with Ant Colony Optimization

Ayah Helal[1,2] | Fernando E. B. Otero[3]

[1]Department of Informatics, King's College London, UK
[2]Department of Computer Science, University of Exeter, UK
[3]School of Computing, University of Kent, UK

Correspondence
Fernando E. B. Otero, School of Computing, University of Kent, Canterbury, CT2 7NF, UK
Email: F.E.B.Otero@kent.ac.uk

Abstract

Data stream mining has recently emerged in response to the rapidly increasing continuous data generation. While the majority of Ant Colony Optimization (ACO) rule induction algorithms have proved to be successful in producing both accurate and comprehensive classification models in non-streaming (batch) settings, currently ACO-based algorithms for classification problems are not suited to be applied to data stream mining. One of the main challenges is the iterative nature of ACO algorithms, where many procedures—e.g., heuristic calculation, selection of continuous attributes, pruning—require multiple passes through the data to create a model. In this paper, we present a new ACO-based algorithm for data stream classification. The proposed algorithm, called Stream Ant-Miner, uses a novel hybrid pheromone model combining both a traditional construction graph and solution archives models to efficiently handle a large number of mixed-type (nominal and continuous) attributes directly without the need for additional procedures, reducing the computational time required to complete an iteration of the algorithm. Our results show that Stream Ant-Miner produces statistically significant concise models compared to state-of-the-art rule induction data stream algorithms, without negative effects on their predictive accuracy.

Ayah Helal and Fernando E. B. Otero equally contributed to this study.

1 | INTRODUCTION

In a data stream context, data arrives in rapid and continuous form as an unbounded sequences of data items. Data stream mining aims at extracting information from the incoming stream of data. While traditional data mining techniques are applied in an off-line mode to discover patterns and/or relationships in data representing useful knowledge, usually characterised by slow data generation where data storage is feasible [1, 2, 3, 4], data stream mining techniques are applied in real-time mode with rapid data generation where data storage is not feasible.

The main properties of data streams are volume, velocity and volatility—these properties present challenges in handling data streams [5]. As data streams volume incrementally increases from zero to potentially infinity (unbounded), data stream mining approaches need to incorporate data in an incremental form without storing all data. Velocity impacts the mining process, preventing the use of any off-line or time consuming procedure due to the fact that data arrives quickly and continuously, and latency of the response is important. Volatility is the concept drift and change of patterns, target, and/or features of the data being mined, which require continuous updates of the model. An example of volatility is the behaviour of customers in an on-line shop, where the prediction of how profitable a week is will differ with the increase of advertising and brand loyalty over time.

Data stream classification techniques are used to find patterns and create predictive models to understand and classify the incoming stream of data [6]. The research in data stream classification has gained attraction due to the importance of its applications and the increase of generation of streaming data [7]. Examples of data stream mining applications include sensor networks [8], web logs [9], computer network [10], and social media data [11]. Krempl et al. [5] highlighted the need to create simpler models, considering not only accuracy but also the interpretability of the knowledge discovered by data stream algorithms. This was one of the recommendations based on the study of real-world applications and the shortcomings of the existing approaches. Notably, current rule induction algorithms in the field follow an incremental approach (e.g., [12, 13]), which leads to large and difficult to interpret models. Ensembles approaches are among the most used used data stream algorithms given their high predictive performance (e.g., [14, 15, 16, 17, 18]), although this is achieved at the expense of comprehensibility as ensembles aggregate the predictions of multiple weak models. Our aim in this paper is to produce accurate, concise, interpretable models to be used in data stream classification.

Classification problems can be viewed as optimization problems, where the aim is to create the best model to represent the predictive patterns in the data. As a result, many metaheuristics have been applied to create classification models, including evolutionary and swarm intelligence algorithms [19, 20, 21, 22]. Ant Colony Optimization (ACO) [23] is amongst the most successful ones. ACO algorithms are inspired by the behaviour of real ants. Ant colonies, and more generally social insect societies, are distributed systems that in spite of the simplicity of their individuals' behaviour present a highly structured social organization. As a result of this organisation, ant colonies can accomplish complex tasks that in some cases far exceed the individual capabilities of a single ant. In ACO, the colony consists of artificial ants, where each ant incrementally builds a solution using the pheromone deposited by other ants and heuristic values. Artificial pheromones are deposited proportionally to the quality of the solution created, guiding the colony towards good regions of the search space.

The first ACO rule induction algorithm for classification problems, called Ant-Miner, was proposed in [24]. Ant-Miner was used to successfully extract *IF-THEN* classification rules from data. Since Ant-Miner introduction, the majority of ACO-based classification algorithms are limited to cope only with categorical attributes, while continuous attributes are discretised in a pre-processing step. *c*Ant-Miner

[25, 26] was the first Ant-Miner extension to cope with continuous attributes directly by employing dynamic discretisation procedure every time a continuous attribute is selected. The discretisation step, either as a pre-processing or dynamic step, is a time-consuming procedure requiring multiple passes through the data as it evaluates candidates threshold values. More recently, Ant-Miner$_{\mathrm{MA}}$ [27, 28] used an archive-based pheromone model to handle continuous attributes directly without a discretisation procedure—this lead to significant gains in terms of computational time, allowing the algorithms to handle larger datasets but showed limitations when there is an increased number of attributes. ACO classification algorithms also employ a pruning procedure, which removes irrelevant attribute-value condition added due to the stochastic nature of the construction process. A pruning procedure requires $n - 1$ passes through the data each time it is used, where n is the number of attributes to be considered for removal. Given the iterative nature of ACO algorithms—i.e., multiple ants create solutions for a number of iterations—these characteristics present a challenge to apply ACO classification algorithms to data streams, where the learning algorithm is required to process data as fast as they arrive and, at the same time, storage of prior data is infeasible and/or not relevant as old data may not represent the current concept.

In this paper, we propose a new approach to extract classification rules from data streams using an ACO-based rule induction algorithm, where the model is replaced instead of incremented to improve its interpretability. ACO algorithms are good candidates for classification problems given that they already have shown good performance in creating classification models, in terms of both predictive accuracy and size of the model—measured as number of discovered rules. While they can handle continuous attributes efficiently without requiring full knowledge of the data distribution [27, 28], a requirement when dealing with data stream, many procedures (e.g., heuristic calculation, pruning) require multiple passes through the data to create a model. The proposed algorithm, called Stream Ant-Miner (*s*Ant-Miner), uses a novel hybrid pheromone model combining both a traditional construction graph and solution archives models to efficiently handle a large number of mixed-type (nominal and continuous) attributes directly without the need for additional procedures, thus making it suitable to handle data streams. We compare the proposed algorithm against state-of-the-art rule induction algorithms for data stream classification using a total of thirteen benchmark data sets. Our results show that the proposed algorithm significantly improve the model size, while competitive in both runtime and predictive accuracy.

The remainder of this paper is organised as follows. Section 2 presents a review of well-known data stream mining algorithms, focusing on rule induction algorithms and Ant-Miner variations. The proposed Stream Ant-Miner (*s*Ant-Miner) algorithm is detailed in Section 3. Computation results are presented in Section 4. Finally, Section 5 presents the conclusions and future research directions.

2 | RELATED WORK

2.1 | Data Stream Classification

Given a stream D of data instances observed over time T, where each data instance is described by a set of predictor attributes' values $\{A_1, \ldots, A_n\}$, a data stream classification problem can be formally specified as the problem of learning a model M over D such that M associates a label $l \in L$ to each instance in D, i.e., $M : D \rightarrow L$. At each time step $t \in T$, the algorithm learns a model M_t to classify the data points in D_t based on the previous observed instances D_{t-1}. Several algorithms for data stream classification have been proposed in the literature—Table 1 presents a summary of data stream classification works.

Hoeffding bound framework [29] is one of the most influential data stream mining techniques. The Hoeffding bound is used in data stream algorithms to obtain probabilistic bounds on the difference between the mean of a distribution and the value of the mean calculated using a subset of the data—i.e., it allows to estimate the mean of a distribution with a probability δ using a subset of the data (finite data). This characteristic fits well within data stream scenarios, where the distribution of the data is unknown (infinite data). Hoeffding bound is generally used to determine number of instances seen by an algorithm to make a decision regarding the model being created. Two of the notable successful algorithms based on this framework are Very Fast Decision Trees (VFDT) and Very Fast Decision Rules (VFDR). VFDT [30] is a data stream decision tree algorithm that does not store any examples in main memory, requiring only space proportional to the size of the tree and associated statistics to calculate the information gain of attributes. For each attribute, VFDT stores the frequency of all seen values. The Hoeffding bound is used to decide on the number of instances needed to be seen at a leaf node before making a decision to create a test and divide the node into two leaves. Computational experiments showed that VFDT using the Hoeffding bound on a subset of the data chose the same node to divide as when the entire data set used. At first, the VFDT algorithm was designed for static data stream without concept drift and provided no forgetting mechanism.

Hulten [31] proposed CVFDT algorithm as an extension of VFDT to cope with concept drift, which uses a fixed window size to determine which nodes in the tree are ageing. Fragments of the tree that become old (after seeing a user defined number of instances) and inaccurate are replaced with alternative subtrees. The process incrementally improves and updates the decision tree, building subtrees to update the model using the data instances of the current window. The resulting accuracy of the CVFDT is similar to what would be obtained by reapplying a VFDT learner to the entire window every time a new instance arrive. Computational experiments showed that CVFDT is better in controlling the size of the tree throughout concept drifts, while VFDT considers many more examples and it is forced to grow larger trees to make up for early decisions becoming incorrect.

Very Fast Decision Rules (VFDR) was proposed by Gama and Kosina [12]. It is a single pass[1] algorithm, similar to VFDT and CVFDT, that learns ordered or unordered classification rules. Similar to VFDT's approach, statistics are stored for all possible values for each attribute. A rule is then expanded with the attribute-value condition (term) that minimizes the entropy of the class labels of the instances covered by the rule. Hoeffding bound is used to determine the number of instances seen before a rule can be expanded or new rule can be induced. Experiments showed that the number of rules produced by VFDR is much smaller than the number of leaves in a tree produced by VFDT.

Stahl et al. [13] proposed the eRules algorithm for rule induction in data streams. eRules uses a fixed sliding window and learns rules using the Prism algorithm [32] on a batch of data (window). Prism is a greedy rule induction algorithm that creates a set of rules, where each attribute-value pair in the antecedent of a rule is chosen to maximise the probability of the target class. New instances are added to a buffer if they are not covered by the current rule set and a majority classifier is used to make a prediction. When the number of instances in the buffer reaches a user-defined threshold, eRules triggers the incremental creation of new rules. To adapt to concept drift, the rule set is validated using the current buffer; if a rule's accuracy is deteriorated as results of mistakenly classifying instances over time,

[1] A single pass algorithm uses each data instance only once.

it is removed from the rule set. eRules use a discretisation procedure, testing multiple intervals to create continuous attribute-value conditions—it is important to note that this process is time consuming.

Le et al. [33] proposed an extension to eRules in order to handle continuous attributes in a more computational efficient way. The proposed G-eRules extension uses a Gaussian distribution on continuous attribute values to efficiently sample values to create continuous attribute-value conditions. The sampling procedure replaces the time-consuming procedure used in eRules. Le et al. [34] added a Hoeffding bound procedure to determine the credibility of a rule term. A rule term is added to the current rule only if the difference of the conditional probabilities between the new rule condition and the second best possible rule condition is greater than the Hoeffding bound.

Xu and Wang [35] proposed the Dynamic Extreme Learning Machine (DELM) for data stream classification. Extreme learning machine (ELM) is a single hidden layer feedback neural network. Due to its fast training and good generalization, ELM has been applied to many fields and recently to data streams. DELM uses two hidden layers to dynamically adjust the ELM layer when concept drift is detected. From the results comparing DELM with different ELM implementations for data streams, DELM achieves a better balance between accuracy and time overhead than online sequence extreme learning machine (OS-ELM) [36]. One of the limitations of models based on neural networks is that they represented black box models that lack comprehensibility. In this paper we focused on interpretable models, represented as a list of *IF-THEN* classification rules.

Several evolutionary approaches have been presented to handle data stream classification. Vivekanandan [19] proposed an online genetic algorithm (OGA), an incremental rule learning algorithm that creates a rule set for data stream classification with concept drift. Each individual is represented as a classification rule, and the algorithm builds the rule set gradually by evaluating each individual on a window of instances, adding the best individual of the population to the rule set. Rules that falls a under user-defined threshold of quality are removed from the rule set. OGA has a limitation of coping with only categorical attributes; continuous attributes are not supported.

Vahdat et al. [20] proposed a GP for streaming data classification tasks with label budgets, where the GP learns a model using a limited number of labelled instances. The approach uses a sampling procedure to select instances from data stream to receive a label and add them to an archive. Then, a GP procedure is triggered to create a model on the archive instances. The best individual of the GP is selected as the anytime classifier. Khanchi et al. [37] proposed an improvement for problems with class imbalance under label budget. The archive and sampling polices are optimised to preform under class imbalanced context, where they incrementally introduce bias in the sampling of the stream and replacement of instances in the archive to balance the classes in the archive and improve the model creation.

Sancho-Asensio et al. [38] proposed a supervised neural constructivist system (SNCS) for mining data streams with concept drift. The SNCS classifier uses a population of multilayer perceptrons (MLP) with feed forward topology (i.e., the signal propagates from inputs toward the output layer). SNCS operates in two modes, the learning mode and the prediction mode. In the learning mode, SNCS discovers and evolves new MLPs that accurately predict a desired label. In the prediction mode, SNCS uses its current knowledge to determine the best label for a new input instances.

Adaptive Random Forests (ARF) [18] is one of the most successful ensemble approaches applied to data stream mining. ARF is an adaptation of the random forest algorithm [39], where each tree is build based on the VFDT algorithm. It employs an adaptive strategy to cope with concept drift: new trees are created once the algorithm detects the possibility of a drift; once the drift is confirmed, these trees replace existing trees on the ensemble. More recently, the popular XGBoost ensemble algorithm [40]

TABLE 1 Examples of data stream mining algorithms.

Algorithm	Model Type	Main Characteristic
VFDT[30]	Decision Trees	Uses Hoeffding bound to decide on the number of instances needed to be seen to split nodes
CVFDT[31]	Decision Trees	Extends on VFDT to use a fixed window size to determine the age of nodes
VFDR[12]	Decision Rules	Similar to VFDT, uses Hoeffding bound to determine the number of instances seen before a rule is expanded
eRules [13]	Decision Rules	Uses a sliding window to learn rules using the Prism algorithm
G-eRules[33]	Decision Rules	Extends eRules by using a gaussian distribution to efficiently sample continuous attributes' values
DELM[35]	Neural Network	Uses two hidden layers to dynamically adjust the learning layer when concept drift is detected
Online GA [19]	Decision Rules	Uses online genetic algorithm that creates rules of each class in parallel
GP [20]	Decision Rules	Uses genetic programming with a sampling procedure for data stream classification under limited label budgets
SNCS[38]	Multi-layer Perceptron	Uses a population of multilayer perceptrons (MLP) with feed forward topology
ARF[18]	Random Forest	An adaptation of the random forest algorithm, where each tree is build based on the VFDT algorithm
AXGB[40]	Decision Trees	Uses an ensemble of weak learners (trees), where older trees are replaced by new trees created on mini-batches of the data

was adapted to classify evolving data streams by Montiel et al. [41]. In Adaptive XGBoost (AXGB), the ensemble is created incrementally by building weak learners (trees) on mini-batches of the data; these learners then replace older learners on the ensemble.

2.2 | Classification with ACO

Ant-Miner was the first ACO-based classification algorithm proposed in the literature [24]. Ant-Miner uses an ACO-based procedure to extract *IF-THEN* classification rules from data. Each rule is a n-dimensional vector of terms t representing tests on specific attribute values that are joined by conjuctions (*AND*s), such that *IF* t_1 *AND* … t_n *THEN predicted value*. Each term t_i consists of a tuple (*attribute*, *operator*, *value*) and they are arranged in a graph structure—each term represents a node in the graph. Ants traverse the graph, selecting terms to create rules. Pheromones are deposited on the edges of the graph indicating sequences of terms that create good rules. A list of rules is created by iteratively creating one rule at a time using the ACO procedure.

Ant-Miner uses a graph-based pheromone model similar to the original Ant-System [42] and the vast majority of Ant-Miner extensions preserved the use of a graph-based pheromone model, focusing on other aspects of the algorithm [43]. Ant-Miner2 [44] and Ant-Miner3 [45] presented a simple heuristic function using density-based estimation; Ant-Miner+ [46] extended Ant-Miner by using a directed acyclic graph to reduce the graph complexity and allow the pre-selection of the predicted class value; *c*Ant-Miner [26] added a dynamic discretisation procedure to handle continuous attribute directly, addressing Ant-Miner limitation of only coping with categorical attributes. A new strategy to create a list of rules, where an ant creates a complete list of rules at each iteration rather than just one rule at a time, was used in *c*Ant-Miner$_{\text{PB}}$ [47]. The idea of pheromone attraction and exclusion was used in Ant-Miner$_{\text{PAE}}$ [48], which extended *c*Ant-Miner$_{\text{PB}}$ pheromone calculation to include weights for attraction and exclusion—ants are attracted by their own pheromone while excluding pheromone from other ants—to improve both exploration and exploitation of the algorithm. Ant-Miner$_{mbc}$ [49] proposed the use of multiple list of rules to create an ensemble, where a weighted voting procedure is used to provide the final classification. AM$_{\text{clr}}$ [50] was proposed as an improvement over Ant-Miner, incorporating a rule evaluation function that takes into account their coverage, length, the number of correctly and incorrectly predicted instances; selection of terms is performed using a rank-based strategy; and it employs a criteria to reject rules that do not meet a minimum threshold of quality and coverage.

Recently, Ant-Miner$_{\text{MA}}$ [27] proposed the use of a pheromone model inspired by ACO$_{\text{MV}}$—an ACO algorithm designed for mixed-variable optimization problems [51]. ACO$_{\text{MV}}$ handles ordinal, categorical and continuous variables using a solution archive as the pheromone model instead of a graph. One of the advantages of adopting a solution archive is the ability to directly cope with different types of attributes, e.g., eliminates the need of a discretisation procedure to handle continuous attribute. Ant-Miner$_{\text{MA}}$ starts by initializing the solution archive with R random generated rules (solutions). Each rule S_i is associated with weight w_i related its quality $Q(S_i)$, where w_i is calculated using a Gaussian function. At each iteration, an ant creates a single rule using an ACO$_{\text{MV}}$ procedure. Ant-Miner$_{\text{MA}}$'s solution archive is used to sample conditions for the creation of the rules, instead of using ants to traverse a construction graph. During rule creation, the algorithm samples the archive to select attributes and corresponding operators and values to create terms to be added to the antecedent of rules. The sampling procedure is based on the type of attribute—either categorical or continuous—and it is influenced by the existing rules on the archive. Once m new rules have been created, where m is the number of ants, they are added into the solution archive. The R and m rules are sorted based on their quality and the m worst ones are removed from the archive. The procedure to create new rules is repeated until a maximum number of iterations is reached. At the end, the best rule (top of the archive) is added to the list of rules. Ant-Miner$_{\text{MA}}$ showed significant improvements in computational time compared to *c*Ant-Miner, one order of magnitude faster in large size data sets, as a result of the elimination of the time-consuming dynamic discretisation procedure.

Previous work identified a limitation of Ant-Miner$_{\text{MA}}$ when the number of attributes increased over fifty, resulting in an increase of the runtime compared to *c*Ant-Miner. Also, *c*Ant-Miner showed improvement in accuracy in data sets where the number of attributes was over fifty, which proved the advantage of the graph pheromone model in selecting the best attributes to use faster than Ant-Miner$_{\text{MA}}$. Ant-Miner$_{\text{MA+G}}$ proposed in [28], combined the graph pheromone model and the archive pheromone model. The proposed approach used a fully configurable framework to automatically design an algorithm. The approach implemented both pheromone model independently and allowed the configurable framework to divide the rule creation between both models. Results showed that such automatically configured design outperformed *c*Ant-Miner to a significant level, indicating that the combination of both pheromone models leads to an improvement in the rule creation procedure. Currently, both pheromone models operate independently, i.e., changes in one model are not automatically reflected on the other. Due lack of integration between the models implementing a learning procedure to create a complete list—such as the one used in *c*Ant-Miner$_{\text{PB}}$—at each iteration would be difficult.

Related works on ACO-based data stream algorithms have mainly focused on clustering [52, 53, 54, 55], an unsupervised[2] learning task where the goal is to group data instances based on their features

[2]Label information for each data instance is not available.

into homogeneous clusters. The only exception, to the best of our knowledge, is the Stream Ant Colony Decision Forrest (strACDF) algorithm proposed in [56] for solving the data stream e-mail foldering problem. Stream ACDF handles data streams by learning an ensemble of classifiers, one classifier for each package of data—a package contains a small number of messages. The algorithm creates *n* classifiers to form the ensemble, where each classifier by itself is a random forest (ensemble) model created by an ACO-based procedure, and additional classifiers can be created by replacing previous ones.

3 | STREAM ANT-MINER: DATA STREAM CLASSIFICATION WITH ACO

In this paper, we propose an ACO-based algorithm for rule induction in data stream classification. The algorithm uses a novel hybrid pheromone model to handle continuous attributes directly without the need for a discretisation step and a simplified construction graph to optimise the selection of attributes. The rationale is to use a construction graph to select attributes and determine their order, which is naturally a combinatorial problem, and a solution archive to select values to create attribute tests, which is a mixed-variable problem. This addresses two current challenges in existing ACO classification algorithms: (1) in graph-based pheromone representation, discretisation of continuous values is time consuming; and (2) in archive-based pheromone representation, large number of attributes affect both the quality and runtime of the algorithm. It is necessary to mitigate these challenges in a data stream context, where both volume and velocity are important aspects of the algorithm.

The proposed algorithm, called Stream Ant-Miner (*s*Ant-Miner), is an anytime prediction data stream algorithm. *s*Ant-Miner uses an ACO-based learning procedure where each ant creates a complete rule list in a single iteration. The continuous learning procedure replaces the classification model (rule list) rather than incrementing it, as illustrated in Figure 1 . The high-level view of the algorithm comprises two procedures. The first procedure is responsible for classifying the incoming data stream (prediction procedure) using a classification model; the second procedure is responsible for updating the classification model by learning from an updated subset of the labelled data (learning procedure). Labelled data, once available, is stored in a buffer. When the buffer is full or a high percentage of misclassification occurs, the learning procedure is triggered to update the model, allowing the algorithm to refine the model and/or adapt to a concept drift.

Considering how the algorithm would operate in real-world scenarios, the unlabelled instances are first classified by the model, then either in an automated or manual fashion, these instances are labelled and fed back to the algorithm to train on them. An example of this is a fraud control system for ecommerce transactions, where a transaction would be first classified by the algorithm, then receive a label (e.g., fraudulent or not) after a period of time and fed back to the model to learn from it. The performance of the algorithm can be monitored by comparing the predicted and real label of transaction.

3.1 | Prediction Procedure

At the very first stage, the classification model used by *s*Ant-Miner is a majority classifier based on the labelled instances and it remains so until the learning procedure is triggered for the first time. Therefore, the initial prediction is a random choice between any of the available class values, then as each

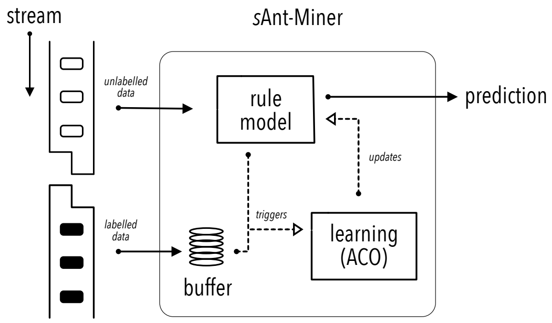

FIGURE 1 Overview of the proposed *s*Ant-Miner: unlabelled data from the stream is classified by the current rule model; the continuous learning procedure is triggered by the availability of labelled data or rate of misclassification to update the current model, allowing the algorithm to refine the model and/or adapt to a concept drift.

labelled instance arrives, a majority classification model is built—the model predicts the most frequent class value to every data instance. When the learning procedure is completed and a new classification model created, the current model is replaced if the new model has a higher predictive quality. Any subsequent prediction is then performed by the new model. Note that in all cases, predictions are very fast even when a classification model different than a majority one is used. Additionally, the model is not incrementally generated as it occurs in algorithms such as VFDT [29], VFDR [12] and G-eRules [33]. Instead, the model is replaced between executions of the learning procedure. Therefore, the model does not suffer from the potential problem of growing indefinitely [12, 13].

3.2 | Learning Procedure

The learning procedure of *s*Ant-Miner uses the buffer of labelled instances and creates a new classifier using an ACO-based procedure. This procedure uses a novel hybrid graph and archive pheromone model, combining the strengths of both: the graph is used to select the attributes to be used while individual archives are used to create rule conditions. As shown in previous work [27, 28], there are advantages in using a hybrid integrated pheromone model, as the archive provides the ability of handling continuous attributes without the need to know the complete attribute values' distribution, while the graph helps to select the best attributes to create the antecedent of rules and their sequence. The learning procedure is triggered when the buffer is full or a high percentage of misclassifications occurs, and it is designed to be fast, running only a limited number of iterations each time it is triggered. If new instances requiring classification arrive before it is completed, the current model performs the classification.

A high-level pseudocode of the learning algorithm is shown in Algorithm 1. At the start of the learning procedure, the current best model (M_{best}) is re-evaluated using the buffer instances (line 2). This is done to assess if a new model (rule list) created during the learning procedure performs better than the current model. Then, for a limited number of iterations, each ant in the colony creates a complete rule list (lines 5-10). The creation of a rule list consists of sampling the current pheromone model to create rules, pruning rules to remove irrelevant attribute conditions, and evaluating the rule list using the buffer instances. Once all ants create a rule list, the iteration-best rule list—i.e., the rule

Algorithm 1: High-level pseudocode of the learning procedure of *s*Ant-Miner.

```
input  : current best model $M_{\mathbf{c}}$, buffer instances
output: rule list model
$M_{\mathbf{best}} \leftarrow M_{\mathbf{c}}$
Evaluate($M_{\mathbf{best}}$, Buffer)
while $t$ < maximum iterations do
    $M_{\mathbf{ib}} \leftarrow M_{\mathbf{best}}$
    while $m$ < colony size do
        $L_m \leftarrow$ CreateRuleList(Buffer)
        Evaluate($L_m$, Buffer)
        if Quality($L_m$) < Quality($M_{\mathbf{ib}}$) then
            $M_{\mathbf{ib}} \leftarrow L_m$
        end
    end
    PheromoneUpdate($M_{\mathbf{ib}}$)
    if Quality($M_{\mathbf{best}}$) < Quality($M_{\mathbf{ib}}$) then
        $M_{\mathbf{best}} \leftarrow M_{\mathbf{ib}}$
    end
    $t \leftarrow t + 1$
end
ClearBuffer()
return $M_{\mathbf{best}}$
```

list with the highest quality among the creates lists—is used to deposit pheromone. If the quality of the iteration-best rule list $M_{\mathbf{ib}}$ is higher than the current best model $M_{\mathbf{best}}$, it replaces $M_{\mathbf{best}}$ as the current best model. Note that when the first iteration does not create a model better than the current best model, $M_{\mathbf{best}}$ is used to update pheromone values until a better performing rule is created. This takes full advantage of how ACO algorithms represent the state of the search: by updating pheromones using the current model, the search is biased to continue around the region of the current model. In other words, the algorithm uses the pheromone as a memory and, as long as there is not a significant decrease in accuracy, the search starts from the same position—this allows the algorithm to refine a model over multiple executions of the learning procedure. The procedure to create a new rule lists (lines 3-16) is repeated until the maximum number of iterations has been reached. Once the maximum number of iterations is reached, all instances in the buffer are removed and the best model is returned. In the case that all rule lists created during the procedure are not better than the current best model, the procedure returns the same model.

Considering the computational time complexity, we divided the analysis into two parts. First, we considered the time complexity of creating a rule list, as this is performed by every ant in the colony. The creation of an individual rule involves the selection of k out of A available attributes. After a rule is created, it undergoes a pruning procedure where the rule is evaluated over D instances to consider the removal of any of its k attributes. Since in the worst case the algorithm could create D rules in a list—one rule per data instance—and select up to A attributes ($k = A$) per rule, the time complexity to create a rule list is $O(A \cdot D^2)$. Second, the creation of a rule list is repeated C (*colony size*) times over

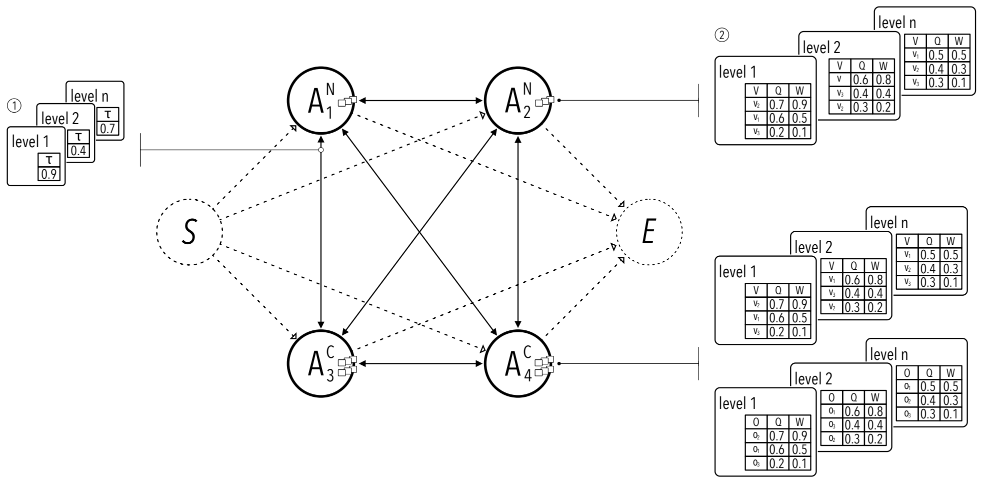

FIGURE 2 Representation of the hybrid construction graph: (1) multiple values are stored on the edges between nodes, representing the likelihood of visiting a node in relation to the level of the rule being created; (2) solution archives are used at each node, one for each level. Note that for categorical attribute nodes (A_i^N), a single solution archive is used to select a value; for continuous attribute nodes (A_i^C), one solution archive is used to select a value and one solution archive is used to select the relational operator; nodes S and E indicate the start and end of the antecedent of a rule, respectively.

I (*maximum number of iterations*), therefore the overall time complexity of the learning procedure is $O(I \cdot C \cdot A \cdot D^2)$. Note that this is the worst case estimation, since the number of data instances D tends to decrease during the creation of rules and k tends to be much smaller than A.

In comparison with the offline (non-streaming) *c*Ant-Miner$_{\text{PB}}$, which also creates complete rule lists in its learning procedure, *s*Ant-Miner improves the rule creation time complexity from $O(A^3 \cdot D^3)$ to $O(A \cdot D^2)$ as the hybrid pheromone model allows the algorithm to efficiently handle continuous attributes and pruning of rules.

3.2.1 | Pheromone Model

The hybrid construction graph consists of a fully connected graph, where each attribute in the data set is represented by a node. Each node holds an archive to sample values for the attribute. An additional archive is used by continuous attribute nodes to sample a relational operator (e.g., “>", “≤"). Two additional nodes are added to represent the start (S) and end (E) of the antecedent of a rule. The pheromone model consists of several levels, which correspond to the indexes of the rules being created (e.g., 1 for the first rule, 2 for the second rule and so forth). This allows the algorithm to store pheromone and archive values for different rules, following a Pittsburgh-style strategy to create rule lists [47]. Figure 2 illustrates the hybrid construction graph and the underlying pheromone model.

Pheromone is deposited on the edges of the construction graph, representing the selected attributes and their order to create rules; the operator and values are sampled from individual archives. Each archive is sorted by the quality of the rule where the entry (*value* or *operator*) appears. The weight of an entry j is calculated by:

$$w_j = \frac{1}{qK\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2K^2}} \tag{1}$$

where q is a user-defined value used to control the extent of the top-ranked entry influence on the sampling of new values, K is the size of the archive and $rank(j)$ is the position of the j-th entry in the archive. The weight of an entry is used during the sampling of new values as an indicator of the level of attractiveness of this value. The greater the weight of an entry, the higher the probability of sampling a new value around it.

3.2.2 | Rule List Creation

When the rule list creation process starts, m ants create rule lists by traversing the construction graph. An ant uses the pheromone model to create multiple rules covering different training instances by specifying a tour identification (level), which corresponds to the index of the rule being created—1 for the first rule, 2 for the second and so forth. This way, each ant will use pheromone entries corresponding to the level of the rule being created during the rule construction process. The pheromone entries are stored in a pheromone matrix, where column and row indexes indicate the edge between two vertices. The probability of an ant to follow the edge leading to a vertex v_j when creating the rule t and located at vertex v_i is given by:

$$P^t_{v_j} = \frac{\tau^t_{v_i,v_j}}{\sum_{k=1}^{\lambda_{v_i}} \tau^t_{v_i,v_k}} \tag{2}$$

where $\tau^t_{v_i,v_j}$ is the amount of pheromone associated with the entry (t, v_i, v_j), and λ_{v_i} is the set of neighbouring vertices of vertex v_i. For each node selected by an ant, a term (*operator*, *value*) is sampled from the node's archive(s) based on the type of the attribute. The rule creation stops when the ant chooses to visit the end node of the graph. At this point, the rule is pruned to remove irrelevant attributes added due to the stochastic nature of the creation procedure, and the value predicted by the rule (consequent) is set to be the majority class value observed among the covered instances—i.e., the instances that satisfy the attribute-conditions represented by the terms in the antecedent of the rule. Finally, the covered instances are removed and another rule is created until the number of instances remaining is equal to or lower than a user-defined threshold of uncovered instances.

The novel hybrid pheromone model in *s*Ant-Miner takes full advantage of the combined construction graph and solution archive approach, using the graph-based approach to select attributes and their order, and solution archives to determine operator and values for terms. Additionally, the rule list creation process automatically learns the minimum number of attributes required by each rule, since the construction graph includes an end (E) node and pheromone is increased between the last attribute node in a rule and the end node. Therefore, every time the pruning procedure removes attributes from a rule, the pheromone update reflects this change. Over time, the creation process will not include irrelevant attributes, relying less on the pruning procedure and thus improving the computational time of the algorithm.

3.2.3 | Archive Sampling

At start of the learning procedure, random values are used since solution archives are empty. Sampling only starts when an archive is full—the size of the archive is determined by a user-defined parameter.

For categorical attributes, the relational operator is always set to EQUAL (=) and a value is sampled from the archive based on ACO$_{\mathrm{MV}}$ categorical value sampling. Consider a categorical attribute i that has N possible values, each value v_l (where $v_l \in \{v_1, v_1, .., v_N\}$) has the probability p_l given by:

$$p_l = \frac{\alpha_l}{\sum_{j=1}^{N} \alpha_j} \tag{3}$$

where α_l is the weight associated to the l-th value of the categorical attribute i. The weight of each value is based on the current values in the archive for attribute i, calculated as:

$$\alpha_l = \begin{cases} \frac{w_h}{u_l} + \frac{q}{\eta}, & \text{if } \eta > 0 \text{ and } u_l > 0, \\ \frac{w_h}{u_l}, & \text{if } \eta = 0 \text{ and } u_l > 0, \\ \frac{q}{\eta}, & \text{if } \eta > 0 \text{ and } u_l = 0, \end{cases} \tag{4}$$

where w_h is the weight of the highest entry calculated according to Equation (1) that uses the value v_l, u_l is the number of entries that use the value v_l, η is the number of values of attribute i that are not present in the archive, and q is a variable that is used to control the extend of the top-ranked entry influence on the sampling of new values. The categorical sampling procedure allows an ant to consider two components when sampling a new value: the first component biases the sampling towards values that are used in high-quality entries but do not occur very frequently; the second component biases the sampling towards unexplored values.

For continuous attributes, each node has two archives: one for the relational operator and another for the value. The relational operator is sampled using a nominal sampling procedure, where the possible values are $\{\leq, >\}$. The value is sampled using a continuous sampling procedure. First, each ant probabilistically chooses an entry j from the archive based on:

$$p_j = \frac{w_j}{\sum_{l=1}^{K} w_l} \tag{5}$$

where p_j is the probability of selecting the j-th entry from the archive to sample the new continuous value around it, K is the size of the archive, and w_j is the weight associated with the j-th entry in the archive calculated according to Equation (1). The new value is sampled using a Gaussian probability density function (PDF), where the mean is set as the value of the selected entry j, as follows:

$$g(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{6}$$

$$\sigma_j^a = \xi \sum_{l=1, j\neq l}^{K} \frac{|S_j^a - S_l^a|}{K-1} \tag{7}$$

where S_j^a is the value of the continuous attribute a in the entry j of the archive, and ξ is a user-defined value representing the convergence speed of the algorithm. Hence, σ_j^a is the standard deviation calculated as the average distance between the values of the attribute a in the entry j and all other entries in the archive.

3.2.4 | Pruning

After the creation of a rule, the rule is pruned using a single pass procedure through the buffer to remove irrelevant rule terms. The pruning procedure calculates the coverage of each rule term based on the instances in the buffer. Then, each term is added to a new rule one at a time in the same order until the quality of the rule decreases or the addition of a term makes the rule cover less than a user-specified minimum number of instances. The remaining unused terms are then discarded.

For the purpose of pruning, the quality of a single rule is measured as *sensitivity* × *specificity*, the same measure employed in Ant-Miner, given by:

$$Q = \frac{TP}{TP + FN} \times \frac{TN}{FP + TN} \tag{8}$$

where TP is the number of covered instances that are correctly classified; FN is the number of instances that are not covered that have the same class as the rule; FP is the number of covered instances that are incorrectly classified; TN is the number of instances that are not covered and do not have the same class as the rule.

3.2.5 | Pheromone update

The pheromone update is divided into two steps: the first step updates the edges of the construction graph, while the second updates each individual archive.

The graph update starts with pheromone evaporation, simulated by decreasing the amount of pheromone of each entry by a user-defined factor ρ. Then, the pheromone value of the entries used in the iteration-best rule list ($M_{\mathbf{ib}}$) is increased based on the quality of the rule list, which corresponds to its predictive accuracy measured on the buffer (training) instances. The pheromone update is given by:

$$\tau_{t,v_i,v_j} = \begin{cases} \rho \times \tau_{t,v_i,v_j}, & \text{if } (t, v_i, v_j) \notin M_{\mathbf{ib}} \\ \rho \times \tau_{t,v_i,v_j} + Q(M_{\mathbf{ib}}), & \text{if } (t, v_i, v_j) \in M_{\mathbf{ib}} \end{cases} \tag{9}$$

where ρ is the user-defined evaporation factor, τ_{t,v_i,v_j} is the amount of pheromone associated with the entry (t, v_i, v_j), t is the tour identification (i.e., the t rule in the rule list), $Q(M_{\mathbf{ib}})$ is the quality of the iteration-best rule list, v_i and v_j are the start and end vertices of the edge, respectively. The values given by Equation (9) are limited to the interval $[\tau_{min}, \tau_{max}]$, following the same approach as the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System (MMAS) [57]. This procedure is the same as the update procedure in the Pittsburgh-based cAnt-Miner$_{\text{PB}}$ [47].

After updating the construction graph edges, each individual archive of a node (attribute) used in a rule is updated. The update consists of adding a pair (*value*, *quality*) to the archive at the level t, where the quality corresponds to the iteration-best rule list quality. Note that for continuous attributes, a pair (*operator*, *quality*) is also added to the operator archive. After all pairs are added, each archive is then sorted based on the pairs' quality and the weight associated with each pair is recalculated based on their (updated) ranks. Finally, the low quality pairs are removed to resize the archive to K pairs.

4 | COMPUTATION RESULTS

The proposed sAnt-Miner was implemented[3] using the Massive Online Analysis (MOA) [58] framework for data stream mining. MOA includes a collection of machine learning algorithms—such

[3]https://github.com/AyahHelal/StreamAntMiner

TABLE 2 Summary of the data sets used in the experiments.

		Attributes		
Data set	# Instances	# Categorical	# Continuous	# Classes
		Real-world		
Airlines	539383	4	3	2
Connect4	67557	42	0	3
Diabetes	101766	34	11	3
Electricity	45312	1	7	2
Forest Type	581012	44	10	7
GMSC	150000	0	10	2
Spam	9831	504	0	2
Poker Hand	829201	5	5	10
		Artificial		
Hyperplane	1000000	0	10	2
LED	1000000	0	24	10
Random RBF	1000000	0	10	2
RT	1000000	5	5	3
SEA	1000000	0	3	2

as classification, regression, clustering, outlier detection, concept drift detection and recommender systems—as well as stream generators and evaluation measures. We used a total of 13 standard benchmark data sets in the evaluation: 6 real-world data sets (Airlines, Connect4, Electricity, Diabetes, ForestType, and PokerHand) from the UCI Machine Learning repository [59]; the "Give Me Some Credit" (GMSC) data set from Kaggle repository[4]; the spam corpus data created as the result of a text mining process on an online news dissemination system [60]; and 5 artificial data sets generated using MOA random data generator. The details of the data sets used on the experiments are shown in Table 2 .

In order to determine optimal values for sAnt-Miner's user-defined parameters, we used the I/F-Race procedure [61]. To maintain the comparison fair, 3 different synthetic data sets were used for tuning—Table 3 presents the details of the data sets used for tuning. The input parameters for I/F-Race and the final selected values are presented in Table 4 .

We compared the proposed sAnt-Miner algorithm against VFDR and G-eRules, both well-known rule induction data stream classification algorithms, and the state-of-the-art Adaptive Random Forests (ARF) ensemble algorithm. VFDR and ARF are available on MOA framework; G-eRules was downloaded from the GitHub repository[5] and upgraded to work on the 20.07 version of MOA. We used the prequential tenfold bootstrap validation with ADaptive WINdowing (ADWIN) [62] evaluation window; ADWIN is a traditional test-then-train evaluation and it has theoretical guarantees that the chosen size is optimal without the need to decide beforehand on the size of the sliding window [63]. For G-eRules, VFDR and ARF, the values over the tenfolds of a single prequential evaluation are averaged since they are deterministic algorithms; for sAnt-Miner the values over the tenfold bootstrap validation with ADWIN of fifteen prequential evaluations (a total 150 evaluations) are averaged to take into

[4]https://www.kaggle.com/c/GiveMeSomeCredit
[5]https://github.com/thienle2401/G-eRules

account the stochastic nature of the algorithm. Following the recommendations in [63], our comparison uses 4 different measures:

1. **Prequential accuracy**: instances are first classified by the algorithm (test) before they are available for the learning procedure (training);
2. **Kappa Statistics**: takes into account the class unbalance of the data stream in three different ways:
   (a) **Kappa**: compares an algorithm's prequential accuracy to a chance classifier (one that assigns the same number of instances to each class as the algorithm being evaluated);
   (b) **Kappa M**: compares an algorithm's prequential accuracy to a simple majority class classifier;
   (c) **Kappa Temporal**: compares an algorithm's prequential accuracy to a persistent classifier (one that predicts the class label of the previous instance for the current instance);
3. **Runtime**: runtime measured in seconds of a single prequential tenfold bootstrap validation with ADWIN evaluation window;
4. **Rules count**: number of rules in model generated.

The Kappa statistics values range from $(-\infty, 1)$[6]; when an algorithm has a Kappa value greater than 0 ($\kappa > 0$), it represents the case where it is more often correct than the baseline (naïve) classifier—$\kappa = 1$ being the extreme case where the algorithm is always correct; a Kappa value less than 0 ($\kappa < 0$) represents the case where the algorithm performs worse than the baseline classifier; and Kappa equal to 0 ($\kappa = 0$) represents the case where the algorithm is correct as often as the baseline classifier.

Tables 5 –10 show the average values of the ADWIN prequential evaluation for each of the different measures—the best value for each data set is shown in bold. In order to measure statistical significance of the differences in algorithms' performance, we used the non-parametric Friedman test with Nemenyi's post-hoc test [65] to compare all algorithms to each other. The last six rows on Tables 5 –10 present the results of the Friedman statistical test, including the algorithm's average rank and adjusted p-value for the pairwise comparison. A symbol ▲ (▼) is used to indicate the case when the average ranking of the algorithm represented by the row is statistically significantly better (worse) than the average ranking of the algorithm represented by the column, corresponding to the case where the obtained p-value is lower than 0.05 ($p < 0.05$).

Table 5 presents the results regarding the prequential accuracy. ARF achieved the highest ranking, with an average rank of 1.23, and it was the most accurate algorithm in 11 out of 13 data sets; *s*Ant-Miner achieved the highest ranking, with an average rank of 2.38, followed by VFDR and G-eRules with 3.08 and 3.31, respectively. Both *s*Ant-Miner and VFDR were each the most accurate algorithm in 1 out of 13 data sets. The statistical test showed that ARF is statistically significantly better than VFDR and G-eRules; no statistical significant differences were observed between the performance of ARF and *s*Ant-Miner.

The results of Kappa measures are presented in Tables 6 –8 —a negative value in these tables indicates that the compared algorithm performs worse than the baseline (naïve) classifier. Table 6 presents

[6]While the exact lower limit depends on the distribution, it is important to note that Kappa statistics can take negative values when the performance of the algorithm is worst than the baseline classifier [64].

TABLE 3 The data sets used in the I/F-Race procedure. All data sets were generated using the MOA data generator.

		Attributes		
Data set	# Instances	# Categorical	# Continuous	# Classes
Wave Form	100000	0	40	3
Sine	100000	0	4	2
Agrawal	100000	3	6	2

TABLE 4 I/F-Race parameters range used in the tuning phase.

Parameters	Range	Final value
Buffer Size	[250, 1500]	1459
Colony Size	[5, 20]	6
ξ	[0.00, 1.00]	0.367
q	[0.00, 1.00]	0.119
Archive Size	[5, 50]	17
Max Iteration	[5, 70]	54
Buffer Trigger	[0.00, 1.00]	0.748
Minimum Cases	[5, 30]	30
Uncovered Percentage	[0.00, 0.2]	0.176

the results regarding the Kappa statistic. Similarly to the prequential accuracy results, ARF achieved the highest ranking (1.31), followed by *s*Ant-Miner (2.38), and both VFDR (3.15) and G-eRules (3.15). It is interesting to note that G-eRules has negative values in the *Poker Hand* and *Hyperplane* data sets, indicating that in these data sets G-eRules performed worse than the chance classifier. The statistical test showed that ARF is statistically significantly better than VFDR and G-eRules; no statistical significant differences were observed between the performance of ARF and *s*Ant-Miner.

Table 7 presents the results regarding the Kappa M measure. Once more, ARF achieved the highest ranking (1.23), followed by *s*Ant-Miner (2.54), VFDR (3.00) and G-eRules (3.23). Note that there are eight imbalanced data sets, where a single class has a high number of instances: *Airlines*, *Connect4*, *Diabetes*, *Electricity*, *Forest Type*, *GMSC*, *Poker Hand* and *Hyperplane*. It is expected the majority class classifier to perform relatively well in these data sets. Comparing the performance of the algorithms against the majority classifier, ARF performs better than the majority classifier in all of them; *s*Ant-Miner preforms better in 6 out of 8 (Kappa M is negative in both *Forest Type* and *Poker Hand*); VFDR performs better in only 2 out of 8 (Kappa M is negative in *Airlines*, *Connect4*, *Diabetes*, *Forest Type*, *GMSC* and *Poker Hand*); and G-eRules performs better in only 2 out of 8 (Kappa M is negative in *Diabetes*, *Electricity*, *Forest Type*, *GMSC*, *Poker Hand* and *Hyperplane*). The statistical test showed that ARF is statistically significantly better than all algorithms—*s*Ant-Miner, VFDR and G-eRules.

TABLE 5 Prequential accuracy

Data set	**G-eRules**	**VFDR**	**ARF**	***s*Ant-Miner**
Airlines	57.92	56.06	**64.66**	59.21
Connect4	69.27	65.67	**77.71**	66.49
Diabetes	50.81	53.83	53.94	**56.06**
Electricity	49.49	70.40	**90.49**	89.18
Forest Type	63.48	64.35	**94.97**	70.30
GMSC	93.32	93.28	**93.54**	93.35
Spam	92.76	82.40	**98.30**	93.00
Poker Hand	53.07	59.58	**89.34**	56.05
Hyperplane	50.00	**71.00**	65.08	69.60
LED	51.28	47.72	**73.97**	47.86
Random RBF	52.14	78.04	**95.18**	68.45
RT	64.58	60.76	**95.44**	57.62
SEA	66.68	80.29	**89.59**	84.02
Average rank	3.31	3.08	1.23	2.38
Adjusted p-values ($\alpha = 0.05$)				
G-eRules	–	0.6486	▼2.4E-4	0.2049
VFDR	0.6486	–	▼0.0013	0.3431
ARF	▲2.4E-4	▲0.0013	–	0.0907
*s*Ant-Miner	0.2049	0.3431	0.0907	–

Bold values indicate the best results per data set.

Table 8 presents the results regarding the Kappa Temporal measure. Similarly to the previous Kappa measures, ARF achieved the highest ranking (1.23), followed by *s*Ant-Miner (2.38), VFDR (3.08) and G-eRules (3.31). There are four data sets with known temporal nature: *Electricity*, *Forest Type*, *Spam* and *Poker Hand*. ARF performs better than the persistent classifier in 3 out of 4 (Kappa M is negative in *Forest Type*); *s*Ant-Miner performs better than the persistent classifier in 1 out of 4 (Kappa M is negative in *Forest Type*, *Spam* and *Poker Hand*); both VFDR and G-eRules perform worse than the persistent classifier in all four data sets. The statistical test showed that ARF is statistically significantly better than VFDR and G-eRules; no statistical significant differences were observed between the performance of ARF and *s*Ant-Miner.

Table 9 presents the results regarding the runtime, measured in seconds. *s*Ant-Miner achieved the highest ranking (1.85), followed by G-eRules (1.92), VFDR (2.23) and ARF (4.00). All three rule induction algorithms are faster than ARF, as expected as ARF build multiple models to create an

TABLE 6 Kappa statistic (percentage)

Data set	G-eRules	VFDR	ARF	*s*Ant-Miner
Airlines	15.27	2.38	**27.77**	13.43
Connect4	22.18	0.00	**45.78**	8.92
Diabetes	2.94	0.04	0.93	**9.63**
Electricity	2.31	41.77	**80.89**	78.39
Forest Type	10.27	13.05	**71.21**	29.41
GMSC	0.00	1.35	**16.57**	10.31
Spam	67.55	18.40	**93.95**	73.09
Poker Hand	-0.02	7.05	**73.68**	7.44
Hyperplane	-0.08	**42.01**	30.17	39.20
LED	45.89	41.95	**71.06**	42.09
Random RBF	4.05	56.08	**90.37**	36.91
RT	40.42	32.93	**92.75**	28.54
SEA	9.42	54.79	**76.83**	64.06
Average Rank	3.15	3.15	1.31	2.38
Adjusted p-values ($\alpha = 0.05$)				
G-eRules	–	1.0000	▼0.0016	0.3862
VFDR	1.0000	–	▼0.0016	0.3862
ARF	▲0.0016	▲0.0016	–	0.1338
*s*Ant-Miner	0.3862	0.3862	0.1338	–

Bold values indicate the best results per data set.

ensemble. The statistical test showed that *s*Ant-Miner, VFDR and G-eRules are statistically significantly better than ARF; no statistical significant differences were observed between *s*Ant-Miner, VFDR and G-eRules.

As aforementioned, the simplicity of the model is also an important evaluation criterion, as discussed in [5]. In order to evaluate the size of the discovered models, we focus on the results regarding the average number of rules, presented on Table 10 . In this case, the lower the number of rules, the simpler (concise) the model is considered. Note that this was not measured for ARF, since ARF builds an ensemble model with multiple decision trees. *s*Ant-Miner creates the smallest models, achieving an average rank of 1.15; VFDR ranked second (2.15) followed by G-eRules (2.69). The results obtained by *s*Ant-Miner are statistically significantly better than both G-eRules and VFDR. This emphasises the advantage of replacing the model between executions of the learning procedure instead of updating (increasing) the same one—as a result, *s*Ant-Miner is able to keep the size of the model relatively small.

TABLE 7 Kappa M statistic (percentage)

Data set	**G-eRules**	**VFDR**	**ARF**	***s*Ant-Miner**
Airlines	2.23	-1.88	**17.67**	5.16
Connect4	10.06	-0.48	**34.77**	1.92
Diabetes	-6.70	-0.15	0.07	**4.68**
Electricity	-7.55	36.98	**79.74**	76.95
Forest Type	-274.71	-227.66	**56.64**	-298.48
GMSC	-0.04	-0.73	**3.30**	0.32
Spam	65.39	15.93	**91.89**	66.57
Poker Hand	-36.75	-17.76	**67.92**	-28.28
Hyperplane	-0.23	**41.87**	29.91	39.06
LED	45.80	41.83	**71.04**	41.99
Random RBF	3.82	55.87	**90.32**	36.61
RT	32.27	24.97	**91.28**	18.97
SEA	6.78	44.86	**70.88**	55.30
Average Rank	3.23	3.00	1.23	2.54
Adjusted p-values ($\alpha = 0.05$)				
G-eRules	–	0.7241	▼4.7E-4	0.5147
VFDR	0.7241	–	▼0.0024	0.7241
ARF	▲4.7E-4	▲0.0024	–	▲0.0392
*s*Ant-Miner	0.5147	0.7241	▼0.0392	–

Bold values indicate the best results per data set.

Overall, the results obtained by the proposed *s*Ant-Miner are positive. It discovered smaller models while maintaining similar predictive accuracy when compared to ARF, considered one of the state-of-the-art data stream classification algorithms; both VFDR and G-eRules rule induction algorithms are statistically significantly worse than ARF. Additionally, *s*Ant-Miner is competitive regarding its runtime, despite being an iterative algorithm. We attribute this to the proposed hybrid construction graph, where a solution archive handles attribute values combined with a construction graph to select attributes to compose rules. This combination addressed the limitation of using only a solution archive to create classification rules, identified in a previous work [27], where using a solution archive did not produce accurate models in data sets with a larger number of attributes (more than 50 attributes). Our results indicate that the hybrid model allows the algorithm to deal with a larger number of attributes effectively: the construction graph focuses on selecting the most relevant attributes and their order, and at the same time, solution archives deal with deciding the values to create attribute conditions; the need for a time-consuming discretisation procedure is replaced by a fast sampling procedure.

TABLE 8 Kappa Temporal statistic (percentage)

Data set	**G-eRules**	**VFDR**	**ARF**	***s*Ant-Miner**
Airlines	4.63	0.46	**19.87**	7.64
Connect4	37.77	30.48	**54.86**	32.13
Diabetes	14.67	19.91	20.09	**23.77**
Electricity	-232.25	-94.69	**37.41**	28.79
Forest Type	-834.40	-800.35	**-26.10**	-645.48
GMSC	48.72	48.38	**50.44**	48.92
Spam	-115.45	-423.29	**49.54**	-108.11
Poker Hand	-77.20	-56.70	**58.47**	-67.45
Hyperplane	0.15	**42.09**	30.09	39.29
LED	45.84	41.88	**71.06**	42.03
Random RBF	4.34	56.11	**90.38**	36.95
RT	43.93	37.88	**92.78**	32.91
SEA	27.44	57.08	**77.34**	65.21
Average Rank	3.31	3.08	1.23	2.38
Adjusted p-values ($\alpha = 0.05$)				
G-eRules	–	0.6485	▼2.5E-4	0.2049
VFDR	0.6485	–	▼0.0013	0.3431
ARF	▲2.5E-4	▲0.0013	–	0.0907
*s*Ant-Miner	0.2049	0.3431	0.0907	–

Bold values indicate the best results per data set.

5 | CONCLUSION AND FUTURE RESEARCH

In this paper, we introduced the ACO-based *s*Ant-Miner algorithm for data stream classification. *s*Ant-Miner uses a novel hybrid pheromone model, combining both construction graph and solution archive pheromone models, to create rule-based classification models. The main motivation is to allow the algorithm to benefit from a solution archive model, which can handle both nominal and continuous attribute values without the need for any preprocessing procedure (e.g., discretisation), and a graph model for selecting the best attributes to use when creating rules. The use of a simplified construction graph, where each attribute is represented by a node, allowed the use of an end node to indicate the termination of the selection of attributes for a rule. Therefore, the algorithm iteratively remembers the (irrelevant) terms removed by the pruning procedure and these are not selected at later iterations, reducing the computational time taken by the pruning procedure. These improvements contribute to the ability of the algorithm to handle large number of attributes and mixed attributes types efficiently, an important aspects in a data stream context.

TABLE 9 Runtime (seconds)

Data set	G-eRules	VFDR	ARF	*s*Ant-Miner
Airlines	2407.67	**172.53**	26222.18	187.72
Connect4	101.52	**13.30**	1695.51	65.40
Diabetes	1248.68	**23.53**	5255.84	82.37
Electricity	**15.13**	45.72	619.53	20.79
Forest Type	1099.44	566.90	18427.74	**538.75**
GMSC	**30.70**	327.65	1723.18	58.28
Spam	88.88	**10.79**	1970.69	138.78
Poker Hand	1570.68	1259.51	14653.76	**505.93**
Hyperplane	**180.85**	2809.90	53431.88	975.91
LED	3009.83	3663.33	38714.37	**2027.74**
Random RBF	**191.29**	6660.39	33207.85	440.59
RT	**874.90**	5874.18	28269.53	1240.41
SEA	**137.65**	2383.27	168649.48	678.19
Average Rank	1.92	2.23	4.00	1.85
Adjusted p-values ($\alpha = 0.05$)				
G-eRules	–	1.0000	▲2.4E-4	1.0000
VFDR	1.0000	–	▲0.0028	1.0000
ARF	▼2.4E-4	▼0.0028	–	▼1.2E-4
*s*Ant-Miner	1.0000	1.0000	▲1.2E-4	–

Bold values indicate the best results per data set.

The proposed *s*Ant-Miner was compared against ARF, VFDR and G-eRules, well-known algorithms from the literature, using a collection of standard benchmark data sets. Our results using 6 real-world and 5 artificial data sets showed that *s*Ant-Miner is competitive in terms of predictive accuracy and data stream oriented evaluation measures; importantly, there are no statistically significantly differences between *s*Ant-Miner and ARF—an ensemble algorithm considered one of the state-of-the-art for data stream mining—in 3 out of the 4 prediction measures. Moreover, the rule list models produced by *s*Ant-Miner were significantly smaller in terms of the number of rules, which contributes to their interpretability since concise models are intuitively easier for an user to visualise and understand their predictions. The results also showed that *s*Ant-Miner is competitive regarding its computational time, despite using an iterative ACO procedure to create the classification model.

There are several interesting directions for future research. In this paper, we tested the performance of the algorithms in data sets without (explicit) concept drift. While it is likely that the real-world data sets do have naturally occurring concept drift, it will be interesting to measure the performance of *s*Ant-Miner in data sets with a known factor of concept drift. Moreover, the novel hybrid pheromone

TABLE 10 Number of rules

Data set	**G-eRules**	**VFDR**	***s*Ant-Miner**
Airlines	2760.63	110.40	**2.47**
Connect4	915.10	19.70	**2.94**
Diabetes	1621.75	13.05	**2.19**
Electricity	368.30	26.90	**3.13**
Forest Type	399.08	34.90	**3.27**
GMSC	315.00	44.00	**1.78**
Spam	181.60	**2.50**	3.32
Poker Hand	1187.38	101.81	**3.28**
Hyperplane	**5.74**	162.64	6.56
LED	121.43	22.13	**8.67**
Random RBF	17.62	203.79	**4.61**
RT	400.45	110.97	**7.64**
SEA	50.22	213.22	**5.11**
Average Rank	2.69	2.15	1.15
Adjusted p-values ($\alpha = 0.05$)			
G-eRules	–	0.1698	▼2.6E-4
VFDR	0.1698	–	▼0.0215
*s*Ant-Miner	▲2.6E-4	▲0.0215	–

Bold values indicate the best results per data set.

model proposed in *s*Ant-Miner had a positive effect in terms of both quality of the rules created and computational time. It would be interesting to evaluate its performance in traditional (offline) classification problems. Moreover, *s*Ant-Miner does not use any heuristic information to guide the search, a common feature in ACO algorithms. The use of heuristic information can have a positive impact on both the quality of the rule lists created and the runtime of the algorithm. Finally, another research direction worth further exploration is to compare the performance of *s*Ant-Miner against interpretable models generated by eXplainable AI systems [66, 67].

References

[1] Fayyad U, Piatetsky-Shapiro G, Smith P. From data mining to knowledge discovery: an overview. In: Advances in Knowledge Discovery & Data Mining. ; 1996: 1–34.

[2] Gama J. *Knowledge Discovery from Data Streams*. CRC Press . 2010. 255 p.

[3] Sayed-Mouchaweh M, Lughofer E. *Learning in non-stationary environments: methods and applications*. Springer Science & Business Media . 2012. 440 p.

[4] Khamassi I, Sayed-Mouchaweh M, Hammami M, Ghédira K. Discussion and review on evolving data streams and concept drift adapting. *Evolving systems* 2018; 9(1): 1–23.

[5] Krempl G, Žliobaite I, Brzeziński D, et al. Open Challenges for Data Stream Mining Research. *SIGKDD Explorations Newsletter* 2014; 16(1): 1–10.

[6] Gaber MM, Zaslavsky A, Krishnaswamy S. Mining Data Streams: A Review. *SIGMOD Record* 2005; 34(2): 18–26.

[7] Ramírez-Gallego S, Krawczyk B, García S, Woźniak M, Herrera F. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing* 2017; 239: 39–57.

[8] Rehman uMH, Chang V, Batool A, Wah TY. Big data reduction framework for value creation in sustainable enterprises. *International Journal of Information Management* 2016; 36(6): 917–928.

[9] Mundhe RV, Manwade KB. Continuous Top-k Monitoring on Document Streams. In: 2018 International Conference on Inventive Research in Computing Applications (ICIRCA). ; 2018: 1008–1013.

[10] Andreoni Lopez M, Mattos DM, Duarte OCM, Pujolle G. Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data. *Concurrency and Computation: Practice and Experience* 2019; 31(20): 1–17.

[11] Aggarwal C, Philip S. Online Analysis of Community Evolution in Data Streams. In: Proceedings of the 2005 SIAM International Conference on Data Mining (SDM). ; 2005: 56–67.

[12] Gama J, Kosina P. Learning Decision Rules from Data Streams. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11). ; 2011: 1255–1260.

[13] Stahl F, Gaber MM, Salvador MM. *Research and Development in Intelligent Systems XXIX (International Conference on Innovative Techniques and Applications of Artificial Intelligence)*ch. eRules: A Modular Adaptive Classification Rule Learning Algorithm for Data Streams: 65–78; 2012.

[14] Minku L, Yao X. DDD: A New Ensemble Approach for Dealing with Concept Drift. *IEEE Transactions on Knowledge and Data Engineering* 2012; 24(4): 619–633.

[15] Baena-Garcıa M, Campo-Ávila dJ, Fidalgo R, Bifet A, Gavalda R, Morales-Bueno R. Early drift detection method. In: . 6 of *Fourth International Workshop on Knowledge Discovery from Data Streams*. ; 2006: 77–86.

[16] Street WN, Kim Y. A Streaming Ensemble Algorithm (SEA) for Large-scale Classification. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ; 2001: 377–382.

[17] Almeida E, Kosina P, Gama J. Random rules from data streams. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM; 2013: 813–814.

[18] Gomes H, Bifet A, Read J, et al. Adaptive random forests for evolving data stream classification. *Machine Learining* 2017; 106: 1469–1495.

[19] Vivekanandan P, Nedunchezhian R. Mining data streams with concept drifts using genetic algorithm. *Artificial Intelligence Review* 2011; 36(3): 163–178.

[20] Vahdat A, Atwater A, McIntyre AR, Heywood MI. On the Application of GP to Streaming Data Classification Tasks with Label Budgets. In: Companion Publication of the Annual Conference on Genetic and Evolutionary Computation. ; 2014: 1287–1294.

[21] Cervantes A, Isasi P, Gagné C, Parizeau M. Learning from non-stationary data using a growing network of prototypes. In: IEEE Congress on Evolutionary Computation. ; 2013: 2634–2641.

[22] Nag K, Pal NR. A Multiobjective Genetic Programming-Based Ensemble for Simultaneous Feature Selection and Classification. *IEEE Transactions on Cybernetics* 2016; 46(2): 499–510.

[23] Dorigo M, Stützle T. *Ant Colony Optimization.* MIT Press . 2004. 319 p.

[24] Parpinelli R, Lopes H, Freitas A. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation* 2002; 6(4): 321–332.

[25] Otero F, Freitas A, Johnson C. cAnt-Miner: An Ant Colony Classification Algorithm to Cope with Continuous Attributes. In: Ant Colony Optimization and Swarm Intelligence (LNCS 5217). ; 2008: 48–59.

[26] Otero F, Freitas A, Johnson C. Handling continuous attributes in Ant Colony Classification algorithms. In: IEEE Symposium on Computational Intelligence and Data Mining (CIDM '09). ; 2009: 225–231.

[27] Helal A, Otero F. A Mixed-Attribute Approach in Ant-Miner Classification Rule Discovery Algorithm. In: Genetic and Evolutionary Computation Conference (GECCO). ; 2016: 13–20.

[28] Helal A, Otero F. Automatic Design of Ant-miner Mixed Attributes for Classification Rule Discovery. In: Genetic and Evolutionary Computation Conference (GECCO). ; 2017: 433–440.

[29] Domingos P, Hulten G. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics* 2003; 12(4): 945–949.

[30] Domingos P, Hulten G. Mining High-speed Data Streams. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ; 2000: 71–80.

[31] Hulten G, Spencer L, Domingos P. Mining Time-changing Data Streams. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ; 2001: 97–106.

[32] Cendrowska J. PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies* 1987; 27(4): 349–370.

[33] Le T, Stahl F, Gomes JB, Gaber MM, Fatta GD. Computationally Efficient Rule-Based Classification for Continuous Streaming Data. In: International Conference on Innovative Techniques and Applications of Artificial Intelligence. ; 2014: 21–34.

[34] Le T, Stahl F, Gaber MM, Gomes JB, Fatta GD. On expressiveness and uncertainty awareness in rule-based classification for data streams. *Neurocomputing* 2017; 265: 127–141.

[35] Xu S, Wang J. Dynamic Extreme Learning Machine for Data Stream Classification. *Neurocomputing* 2017; 238(C): 433–449.

[36] Liang NY, Huang GB, Saratchandran P, Sundararajan N. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks* 2006; 17(6): 1411–1423.

[37] Khanchi S, Heywood MI, Zincir-Heywood AN. Properties of a GP Active Learning Framework for Streaming Data with Class Imbalance. In: Genetic and Evolutionary Computation Conference. ; 2017: 945–952.

[38] Sancho-Asensio A, Orriols-Puig A, Golobardes E. Robust on-line neural learning classifier system for data stream classification tasks. *Soft Computing* 2014; 18(8): 1441–1461.

[39] Breiman L. Random Forests. *Machine Learning* 2001; 45: 5–32.

[40] Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). ; 2016: 785–794.

[41] Montiel J, Mitchell R, Frank E, Pfahringer B, Abdessalem T, Bifet A. Adaptive XGBoost for Evolving Data Streams. In: 2020 International Joint Conference on Neural Networks (IJCNN). ; 2020: 1–8.

[42] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 1996; 26(1): 29–41.

[43] Martens D, Baesens B, Fawcett T. Editorial survey: swarm intelligence for data mining. *Machine Learning* 2011; 82(1): 1–42.

[44] Liu B, Abbass HA, McKay B. Density-based heuristic for rule discovery with ant-miner. In: Australasia-Japan Joint Workshop on Intelligent and Evolutionary Systems (AJWIS-2002). ; 2002: 180–184.

[45] Liu B, Abbass HA, McKay B. Classification rule discovery with ant colony optimization. In: IEEE/WIC International Conference on Intelligent Agent Technology. ; 2003: 83–88.

[46] Martens D, De Backer M, Haesen R, Vanthienen J, Snoeck M, Baesens B. Classification With Ant Colony Optimization. *IEEE Transactions on Evolutionary Computation* 2007; 11(5): 651–665.

[47] Otero F, Freitas A, Johnson C. A New Sequential Covering Strategy for Inducing Classification Rules With Ant Colony Algorithms. *IEEE Transactions on Evolutionary Computation* 2013; 17(1): 64–76.

[48] Yang L, Li K, Zhang W, Ke Z. Ant colony classification mining algorithm based on pheromone attraction and exclusion. *Soft Computing* 2017; 21(19): 5741–5753.

[49] Liang Z, Sun J, Lin Q, Du Z, Chen J, Ming Z. A Novel Multiple Rule Sets Data Classification Algorithm Based on Ant Colony Algorithm. *Applied Soft Computing* 2016(38): 1000–1011.

[50] Ayub U, Ikram A, Shahzad W. AM$_{\mathrm{clr}}$: An Improved Ant-Miner to Extract Comprehensible and Diverse Classification Rules. In: Genetic and Evolutionary Computation Conference. ACM; 2019: 4–12.

[51] Liao T, Socha K, Oca M. dM, Stützle T, Dorigo M. Ant Colony Optimization for Mixed-Variable Optimization Problems. *IEEE Transactions on Evolutionary Computation* 2014; 18(4): 503–518.

[52] Fernandes C, Mora A, Merelo J, Rosa A. KANTS: A Stigmergic Ant Algorithm for Cluster Analysis and Swarm Art. *IEEE Transactions on Cybernetics* 2014; 44(6): 843–856.

[53] Fahy C, Yang S, Gongora M. Finding Multi-Density Clusters in non-stationary data streams using an Ant Colony with adaptive parameters. In: IEEE Congress on Evolutionary Computation. ; 2017: 673–680.

[54] Fahy C, Yang S, Gongora M. Ant Colony Stream Clustering: A Fast Density Clustering Algorithm for Dynamic Data Streams. *IEEE Transactions on Cybernetics* 2019; 49(6): 2215–2228.

[55] Lucky L, Girsang AS. Hybrid Nearest Neighbors Ant Colony Optimization for Clustering Social Media Comments. *Informatica* 2020; 44(1): 63–74.

[56] Kozak J, Juszczuk P, Probierz B. The hybrid ant colony optimization and ensemble method for solving the data stream e-mail foldering problem. *Neural Computing and Applications* 2020; 32(19): 15429–15443.

[57] Stützle T, Hoos HH. MAX-MIN Ant System. *Future Generation Computer Systems* 2000; 16(9): 889–914.

[58] Bifet A, Holmes G, Kirkby R, Pfahringer B. MOA: Massive Online Analysis. *Journal of Machine Learning Research* 2010; 11: 1601–1604.

[59] Dua D, Graff C. UCI Machine Learning Repository. 2019. [http://archive.ics.uci.edu/ml].

[60] Katakis I, Tsoumakas G, Banos E, Bassiliades N, Vlahavas I. An adaptive personalized news dissemination system. *Journal of Intelligent Information Systems* 2009; 32(2): 191–212.

[61] López-Ibáñez M, Dubois-Lacoste J, Caceres LP, Stützle T, Birattari M. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 2016; 3: 43–58.

[62] Bifet A, Gavalda R. Learning from time-changing data with adaptive windowing. In: SIAM International Conference on Data Mining. ; 2007: 443–448.

[63] Bifet A, Francisci Morales dG, Read J, Holmes G, Pfahringer B. Efficient Online Evaluation of Big Data Stream Classifiers. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ; 2015: 59–68.

[64] Cohen J. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 1960; 20(1): 37–46.

[65] Demšar J. Statistical Comparisons of Classifiers over Multiple Data Sets. *Machine Learning Research* 2006; 7: 1–30.

[66] Nascita A, Montieri A, Aceto G, Ciuonzo D, Persico V, Pescapé A. XAI meets Mobile Traffic Classification: Understanding and Improving Multimodal Deep Learning Architectures. *IEEE Transactions on Network and Service Management* 2021. doi: https://doi.org/10.1109/TNSM.2021.3098157

[67] Hagras H. Toward human-understandable, explainable AI. *Computer* 2018; 51(9): 28–36.