An Online Algorithm for Detecting Anomalies using Fuzzy Clustering

Darrin M. Hanna[1], Michael F. Lohrer[1], David E. Stern[1], Alexander Postlmayr[1], Adam Kollin[2], Shuo Wang[3], and Gang-yu Liu[3]
[1]Department of Electrical and Computer Engineering, Oakland University, Rochester, MI, United States
[2]RHK Technology, Inc., Troy, MI, United States
[3]Department of Chemistry, University of California Davis, Davis, CA, United States

Abstract—*A fuzzy clustering algorithm with the ability to learn unsupervised can be used to detect objects of interest in semi-structured data. An online application of a fuzzy clustering algorithm with merging was implemented in both software and hardware to test anomaly detection in atomic force microscopy (AFM). The requisite components of the algorithm were all estimated, measured, and verified to meet real time constraints for incoming data. After clusters have been formed, representing the background of an image, any new cluster is an abnormality to the surface which is of interest to the user. This real-time detection of anomalies is important for identifying regions of interest for faster and higher resolution scanning. Results show this application is successfully capable of detecting anomalies in AFM topographic images. The approach taken in this paper is generic and can be applied to other applications with a continuous data stream.*

Keywords: Fuzzy clustering, anomaly detection, atomic force microscopy

1. Introduction

With a growing number of applications ranging from security to computer vision, there is an increasing demand for the ability to identify anomalies in a continuous data stream [1], [2]. The literature in sequential data modeling is predominantly time-dependent where far fewer publications exist for time-independent data, such as [3]. Anomalies among periodic and structured data are generally easy to identify while detecting an irregularity among unstructured, aperiodic data is more difficult. For applications where data are streaming and the anomaly detection is used in control loops or in embedded systems such as network routers and firewalls, the anomaly detection must happen in embedded hardware with constraints on resources and power while operating at high speeds. Further, while identifying irregularities in data streams, the algorithm must be able to adapt to the changing environment where an anomaly detected at one point in time may become a regular occurrence in the longer run.

In this paper, we describe a clustering method in which a set of clusters are formed to represent patterns in data and to identify when anomalies occur. Clustering algorithms have several obstacles when used in online applications. In general, many clustering algorithms require that the number of clusters is specified a priori and must be re-clustered at a great computational expense in order to train from online data. Accordingly, foundational clustering algorithms such as c-means and k-means clustering where the number of clusters must be known in advance [4] are not conducive to applications that have an unknown number of regular patterns. Additionally, the re-clustering of recurring patterns can be extremely costly.

High-dimensional data streams are particularly challenging [5], [6]. Many modifications for clustering algorithms have been created for such online applications aimed at organizing high-dimensional data in real time. In previous work such as [7] a method is presented that is able to detect anomalies in a continuous data stream while adapting over time without having to specify the number of patterns/clusters a priori. The fuzzy clustering algorithm described in the paper uses an efficient merging method that allows for new clusters to be merged or added to the global cluster set every iteration without having to re-cluster. This paper proposes an embedded solution to online learning with streaming data that can:

1) Detect anomalies in real-time
2) Adapt over time to changing data
3) Operate directly in hardware using an FPGA

We implement this method in an FPGA given tight timing constraints to perform anomaly detection on an image taken with atomic resolution while the image is being acquired from an atomic force microscope.

1.1 Atomic Force Microscope Scan Data

We apply our algorithm to a data stream of topographical data sampled continuously from an atomic force microscope (AFM). Examples of anomalies that are of interest in an AFM scan include static deformations or height differences

This material is based upon work supported by the National Science Foundation under Grant No. 1721926.

in the soft surface and dynamic surface reactions taking place on the sample. The benefit of identifying the anomalies in real-time is to change the scan characteristics from a broad sample scan to one that is limited to the specific location where the deformation is or reaction is taking place and scanning at a higher scan speed and resolution. Since this detection is part of the scan control loop it must be implemented in hardware to minimize delay between detecting a dynamic surface reaction, for example, and changing the scan characteristics to capture it. Depending on the surface of interest, the scan data ranges in periodicity and structured; a smooth gold surface, for example can be relatively periodic and structured while a rough gold surface is semi-structured and lacks periodicity making identifying anomalies difficult.

2. System Specifications Outline

Scan probe microscopy (SPM) controllers control scan probe movement and acquire data related to the samples' topology in a high speed control loop in hardware. These data are typically sent through an Ethernet connection to a computer where software processes the data, assembling them into an image. In order to accomplish our objective, once an anomaly is detected, the probe positioning, scan window, and scan speed must be altered as quickly as possible to start scanning the area of interest. Speed is particularly important if the area of interest is a surface reaction that is progressing quickly.

We developed our system based on a combination of low-noise electronics from RHK Technologies' R9 SPM controller, a well-known leading controller for nano-imaging, and custom hardware developed at Oakland University. [8].

The software (R9S) runs on a PC and communicates with the controller through an Ethernet connection. The PCs have more than enough processing power to accommodate running the feature finding algorithm. However, the feature finding system will require reading data from the controller and sending data back to update the scan. Sending data from the R9 to the PC and back adds latency. The runtime of the feature finding algorithm in software combined with the Ethernet latency would mean a response time would not be guaranteed. Since our system is a control loop, variable latency times could cause major issues, thus a software implementation was not chosen. The impact of variable latency would mean the application is no longer real-time.

We use Field Programmable Gate Arrays (FPGAs) to process the data being sent and received in the system. The hardware on the FPGAs has access to all the data and has no extra communication latency with the incoming samples, but are limited by space. The FPGA that would be used for feature finding is already 84% full with other hardware necessary for the standard scan system. Although limited hardware space is available, it is estimated by synthesis tools that the feature finding algorithm can be designed to fit in the remaining space.

Algorithms implemented on FPGA hardware are inherently more difficult to design and verify. Therefore, a software implementation was created to simulate the hardware. Quickly prototyping algorithm changes is possible in the software simulation and verifying correctness is much easier. Finally, the hardware results were compared to the software results to verify the hardware's accuracy.

3. Fuzzy Feature Finding

There are three main components of the fuzzy clustering algorithm with merging that make it ideal for detecting anomalies in continuous data streams: generating features, clustering without a predetermined number of clusters, and merging clusters without re-clustering all data. The features generated are specific to the application for which they are being used. After generating features, the fuzzy clustering algorithm creates clusters in feature space. Lastly, the new clusters are merged with the existing clusters using a resource-saving method that does not require all data to be re-clustered.

3.1 Generating Features

Once a segment of n samples has been received, the fuzzy clustering algorithm generates features for each sample. The features are used for characterizing the clusters to be calculated in feature-space. For each sample, p features are generated to create a $p \times n$ matrix P.

3.2 Clustering Without a Predetermined Number of Clusters

An important aspect of this fuzzy clustering algorithm is its ability to create clusters without prior knowledge of how many global clusters should exist in the data. This is achieved through multiple manipulations of the generated features. After the features have been generated, an $n \times n$ fuzzy compatibility relation matrix Q is calculated to find the similarity of each sample to every other sample, where $x \in P$.

$$q_{ij} = 1 - \frac{1}{p}\left(\sum_{k=1}^{p}\left|\frac{x_{ik} - x_{jk}}{M_k - m_k}\right|^s\right)^{\frac{1}{s}} \quad (1)$$

For feature k, M_k and m_k represent the maximum and minimum values respectively so $|M_k - m_k|$ is that feature's range. Dividing each feature difference by the range normalizes the data and prevents any of the features from having a larger impact than the others and also reduces the effects of noise in the data. The norm calculated is set by s where $s \geq 1$. The Manhattan distance between features is calculated when $s = 1$ and $s = 2$ calculates the Euclidean distance.

As shown in [9], creating clusters involves taking α-cuts of the fuzzy equivalence relation matrix; the fuzzy equivalence relation is calculated by taking the transitive closure of the fuzzy compatibility relation matrix. Using the

max-min transitive closure The fuzzy equivalence relation matrix is transitive if every entry meets the condition shown in Equation 2. Since the transitive closure does not modify the size of the matrix, its result T will also be an $n \times n$ matrix where n is the number of samples.

$$t_{ij} \geq \max_{1 \leq k \leq n} \min\left(t_{ik}, t_{kj}\right) \tag{2}$$

The result of the transitive closure T is compared to α to create a set of clusters C. α is the clustering threshold and has a range of [0,1]. An α closer to zero means sample data will form fewer, larger clusters. An α closer to one means sample data will form more, smaller clusters.

$$\forall i, j \in C_k, t_{ij} \geq \alpha \tag{3}$$

Next, local clusters are created using Equation 4. Using this method, a new local cluster could be formed for every single data point analyzed, or all of the data in the sample set could be grouped into the same cluster.

$$a_i = \frac{1}{|C_i|} \sum_{j \in C_i} x_j \tag{4}$$

The centroids, A, representing the set of clusters are calculated using Equation 4. $|C_i|$ is the cardinality of that specific cluster set and x_j represents the data points from which the cluster sets are comprised.

3.3 Merging Without Re-clustering All Data

The last step of the algorithm is to merge the newly created clusters into the set of existing, global clusters. This step is important because if a new cluster is formed during merging, a new pattern has been detected. Merging occurs every time new clusters are calculated, but does not take more time to calculate because it is accomplished in a way that does not require the re-clustering of all the data. First, a fuzzy compatibility relation is found between the newly created clusters and the existing clusters using Equation 5. The fuzzy compatibility relation here is done for the same reason it was done when creating clusters.

$$z_{ij} = 1 - \frac{1}{p}\left(\sum_{k=1}^{p} \left|\frac{a_{jk} - b_{ik}}{r_k}\right|^s\right)^{\frac{1}{s}} \tag{5}$$

a_{jk} is a data point coming from the set of local clusters found belonging to cluster set A and b_{ik} is a data point from the set of global clusters belonging to cluster set B. The range of the feature across both data sets is r_k.

After computing the relation, the maximum of each row is found and their columns are recorded. Each row represents the distances between a single cluster in A and all clusters in B. Each maximum is then compared to β, the merging threshold that has a range of [0,1]. If the row's maximum is less than β, the cluster that row represents does not merge with any global clusters. Instead, the local cluster becomes a new global cluster.

$$\rho_i^B = \max_j z_{ij} \tag{6}$$

If the row's maximum is greater than or equal to β, the local cluster is merged with the global cluster using Equation 7.

$$\left.\frac{w_{aj}a_j + w_{bi}b_i}{w_{aj} + w_{bi}}\right| z_{ij} = \rho_i^B \tag{7}$$

The weights of the cluster set's features are w_{aj} and w_{bi}. These represent how many clusters are in those data sets. When merging, ρ_i^B replaces the entry in B, thus updating the global cluster set.

A new, non-merged global cluster represents a pattern in the data stream that has not been seen before. When scanning the first few sets of sample data in our application, new global clusters do not signify something has been found. Instead, the first new global clusters represent the background of an AFM scan, or what most of the image should look like. The number of iterations the algorithm goes through for gathering background data could be played around with by the user and optimized for different applications. After scanning the background and generating its related clusters, any new global cluster represent an objects of interest.

4. Software Implementation

Large hardware designs are much harder to verify than the equivalent software design. Thus, software was developed to simulate the hardware design and allow testing and verification of the algorithm. Unlike a standard implementation of fuzzy clustering which would likely use floating-point arithmetic, fixed-point arithmetic was used to be hardware-friendly. Other optimizations for hardware such as a pipeline are implemented and are in their respective sections.

4.1 Sampling

The data points are received individually as 21-bit fractional numbers ranging $[-1, 1-2^{-20}]$ from the microscope. This fixed-point format is Q0.20, read as a signed 21-bit number with a sign bit, no integer bits, and 20 fractional bits. To form clusters, 16 samples must be grouped into an individual segment. This number is chosen as a trade off between robustness of the algorithm and computational power. A higher number of samples in each cluster reduces the susceptibility to noise, on the other hand, the transitive closure bottleneck of the algorithm would decrease throughput and increases the latency of the component pipeline. A power of 2 length is ideal for hardware memory accesses; therefore, the optimal segment length for this application was found to be 16, where 32 samples in a segment is not possible using a real-time application of the hardware.

In addition to the 16 samples that form a segment, 16 additional samples are stored to be used in feature generation.

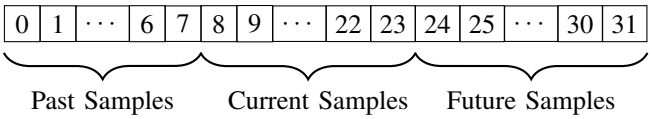

Fig. 1: Data Samples as they are passed into the Feature Generator Component.

The eight samples previous and the eight following samples are saved as well. Requiring eight samples beyond the active segment means adding a latency of eight samples, but this is acceptable to our application as the latency is insignificant. Further discussion can be found in the hardware section of this paper. Every time 16 new samples are recorded, the 32 total samples get passed to the feature generator as shown in Fig. 1. Using the eight past and future samples allows bidirectional features to be extracted. Whereas, the one-dimensional data stream using segments to compute the clusters from memory is preserved.

4.2 Feature Generation

In order to detect a surface anomaly, it is expected that at some locations a sample will have a drastically different value than nearby neighbors. For this reason, the features used in this application are mostly one-dimensional differences between samples' neighbors and the values of the samples themselves. Simple features were chosen to reduce the hardware implementation's resource usage and increase its speed. One-dimensional features were used to limit how many samples needed to be stored in memory; there is not enough memory available in the FPGA to store multiple rows of data for two dimensional feature capabilities. It should be noted that this would drastically increase latency from the existing eight data points to multiple rows of the scan. Currently, the implementation for this algorithm is for images created from a raster scan; however, by using one-dimensional features, the functionality of the algorithm can be kept the same while using different scan patterns such as a Lissajous curve.

With the 16 samples of each segment, eight features are generated:

1) The sample itself.
2) The sample minus the previous sample.
3) The sample minus three samples previous.
4) The sample minus six samples previous.
5) The sample minus eight samples previous.
6) The sample minus the next sample.
7) The sample minus three samples ahead.
8) The sample minus eight samples ahead.

Since each sample is 21 bits, the features are also kept as 21-bit fixed-point numbers in Q0.20 format. Thus, the features selected are easy to compute.

4.3 Compatibility Relation

The first step in the fuzzy compatibility relation is to calculate the range of each feature over the entire segment. By dividing each feature by the range of the feature, the segment is normalized. However, performing a division in hardware is relatively expensive. To eliminate the division, a method that divides by powers of two is developed. A division by a power of two is simply a bit shift, which requires much less time to execute in hardware.

To avoid a square root operation, Manhattan distance is used so s is set to 1. Equation 1 with bit shifts and $s = 1$ becomes

$$q_{s\_ij} = 1 - \frac{1}{p}\sum_{k=1}^{p} 2^{-\lceil \log_2(M_k - m_k)\rceil}|x_{ik} - x_{jk}| \qquad (8)$$

where Qs is the bit shifting version of the fuzzy compatibility relation. Since the number of features p is a power of 2, all divisions have been reduced to bit shifts. Note that $Qs \leq Q$ since $2^{-\lceil \log_2(M_k - m_k)\rceil} \leq (M_k - m_k)^{-1}$. Thus the α parameter should be set lower when using this approximation. This shifting method was also implemented for the compatibility relation done in the merging portion of the algorithm. For that compatibility relation, the β parameter should also be set lower when using this approximation.

4.4 Transitive Closure

The transitive closure is an expensive operation to compute, so optimizations were necessary. Using the fact that a fuzzy compatibility relation is reflexive, the diagonal elements are all ones and will remain as ones for the output of the transitive closure. Also, fuzzy compatibility relations are symmetric so the elements $q_{s\_ij} = q_{s\_ji}$ for all i and j. Therefore only the lower or upper triangular part needs to be computed. With these modifications the transitive closure computation time is approximately halved. Additionally, the Floyd-Warshall algorithm was used to reduce the worst-case time complexity, as detailed in Algorithm 2 in the Hardware Implementation section.

4.5 Clustering

The result of the transitive closure is used to determine which samples belong to a fuzzy equivalence class and therefore get clustered together. Instead of computing a binary matrix representing the fuzzy equivalence classes and then computing centroids from that, the fuzzy clustering algorithm has been implemented such that only a single pass is required. Meaning that the α cut is taken iteratively. The algorithm is given as Algorithm 1, with T the result of the transitive closure, P the feature matrix, n the number of samples to cluster, p the number of features, and α the clustering threshold parameter.

Algorithm 1 Clustering from Fuzzy Relation

Input: Matrix T, Matrix P, Dimension n, Dimension p, Parameter α
Output: Clusters C, Weights w

```
g_i ← 0, i = 1,...,n                ▷ All samples unclustered
m ← 1                               ▷ Current cluster
for i = 1,...,n do
    if g_i = 0 then
        w_m ← 0
        f_j = 0, j = 1,...,p
        for j = 1,...,n do
            if q_ij ≥ α then
                g_j ← 1
                w_m = w_m + 1
                for k = 1,...,p do
                    f_k = f_k + p_kj
                end for
            end if
        end for
        for k = 1,...,p do
            C_km = f_k / w_m
        end for
        m ← m + 1
    end if
end for
```

4.6 Merging

In the fuzzy clustering algorithm, an existing global cluster is updated in the merging process using Equation 7. This calculation was reduced to using one multiplication by transforming it into Equation 9.

$$\left(\frac{w_{bj}}{w_{aj}+w_{bi}}\right)\left(b_i - a_j\right) + a_j \Big| z_{ij} = \rho_i^B \tag{9}$$

Making this optimization to the algorithm saved enough clock cycles to make the component meet the timing requirement and it requires less space in hardware. These specifications will be touched upon in further detail in the Hardware Implementation section.

4.7 Software Implementation Results

A scan of gold [10] was converted from a 512x512 pixel image to a data stream that simulates a raster scan. This data was fed to the algorithm the same way an SPM controller would receive it. Anomalies detected have been marked on the image in Fig. 2 with a dotted black and white cross. The first thirty data were used for background training. Any global clusters created after the training data were flagged.

The first white circle in the top-right of the image is the first anomaly detected. The next small circle on the left side of the image is not flagged. This is expected because the algorithm has already analyzed something similar, the first anomaly. The second anomaly detected is the larger surface deformation in the bottom-left quadrant of the image. This was detected because a deformation that large had not been seen by the algorithm yet. These are the optimal results because the two surface deformations that were expected to be found were detected.

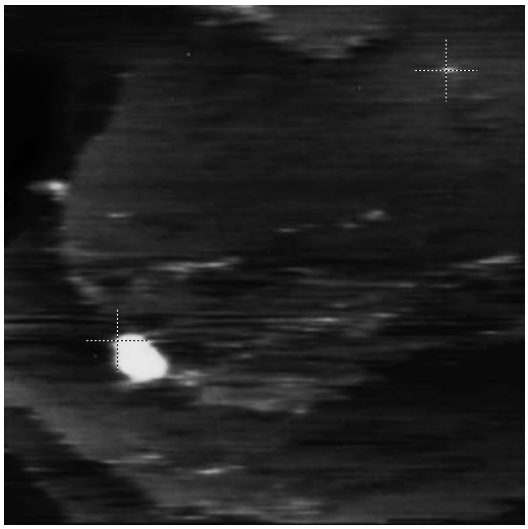

Fig. 2: Topographic scan of Au(111). The α-cut is set to 0.75 and β is set to 0.65. For the fuzzy equivalence relation, the division of the range was used, fixed point math was used instead of floating-point, and the number of possible global clusters was limited to 16.

When using the bit-shifting method, α and β may need to be lowered in order to achieve the same results as the division method. By lowering α to 0.70 from 0.75 and β from 0.65 to 0.56, the results were reproduced. This shows that the bit-shifting method is capable of producing the same optimal results as the division method for Fig. 2.

The final global cluster data resulting from simulating Fig. 2 was compared to results using floating-point. All other parameters for the simulation were kept the same. As expected, the results had slight differences since data was truncated in the fixed point version. The absolute relative difference between the data sets, including both cluster features and weights, was below 1.5%. For both versions, the same two anomalies were detected at the same location. None of the clusters for the anomalies were merged differently between the two versions and less than 0.02% of clusters merged differently for the background data. These results show the same results are achieved using the fixed point method without needing to alter parameters.

A critical aspect of implementing the merging portion of the algorithm was determining the maximum number of possible global clusters. Due to a finite amount of memory in hardware, a limit had to be set to prevent unbounded growth. Under a well tuned alpha and beta, such as those used in Fig. 2, the number of clusters was found to not exceed 16. Only

four global clusters were created to describe the background data when simulating Fig. 2 and two clusters were used when the anomalies were detected. For the application of AFM scanning, it is not expected to need more than a couple clusters for anomalies because action will be taken after discovering one such as setting up a new focused scan. Due to the limited number of clusters needed in this example and others similar to it that were tested, it was determined that limiting the number of clusters was acceptable.

5. Hardware Implementation

Numerous restrictions shaped the design of the hardware implementation. The main restriction was to make sure the hardware could perform the fuzzy clustering algorithm online. Given the specifications of the our system in Table 1, the algorithm must have a run time under 8,000 clock cycles.

One of the methods used for reaching this goal was breaking the algorithm into parts that could be pipelined. A result of this is each component has to operate in under 8,000 clock cycles instead of the entire algorithm reaching this timing goal. The algorithm was broken into the following parts: generate features, find fuzzy equivalence relation, compute transitive closure, create clusters, and merge new clusters into existing clusters. By breaking the algorithm into five components, there is a latency of four iterations. For our application this equates to 32,000 clock cycles, or a 320 μs latency.

The top-level implementation of the hardware is displayed in Fig. 3. Each of the components are given a signal to start every 8,000 clock cycles. As samples enter the R9, they are routed to the Sample Collector component. The Sample Collector stores data sequentially in memory until a full sample set is collected, at which point a flag is raised signaling the data is ready. When a full data set is ready and the start signal is high, the Feature Generator calculates the features. The features are then stored in P, a set of four RAMs. The P RAM that the features get stored in is rotated sequentially every iteration. There are four of these RAMs because the Feature Generator is the first component in the pipeline and the features are needed to create the clusters, which is three stages down the pipeline. Then, the Compatibility Relation uses the data most recently stored in P to compute the compatibility relation, which is then stored in Q. Q is also a set of four RAMs; although it is one stage farther down the pipeline, the Transitive Closure component loads the data from Q and then stores it back

Table 1: System Specifications

Incoming Data Speed	200 kHz
R9 FPGA Clock Speed	100 MHz
Data Points in a Sample Set	16
Clock Cycles per Sample Set	8,000

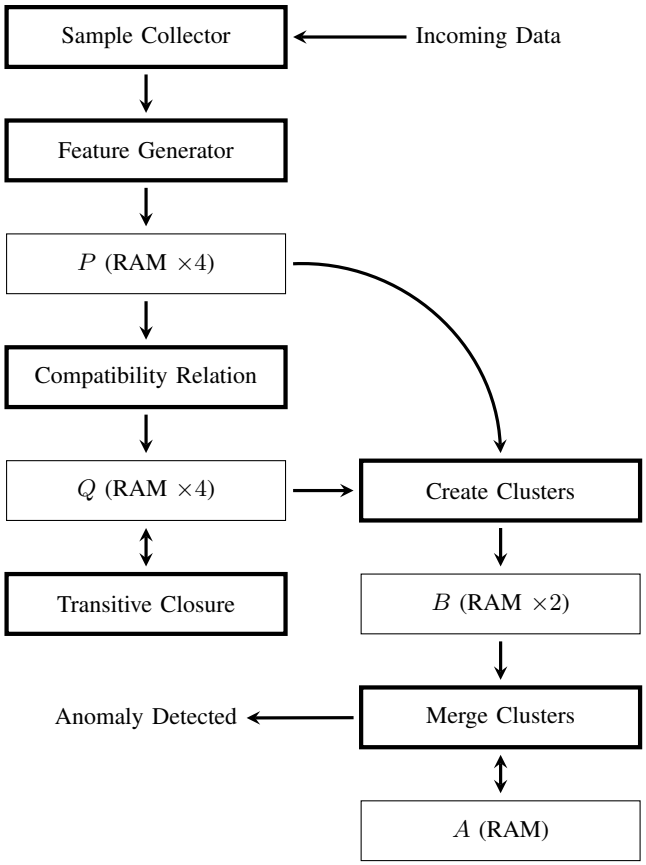

Fig. 3: Top-level diagram of the hardware implementation. Components that have an arrow directed toward a RAM store an output in that RAM. Components with an arrow directed towards it from a RAM use data stored in that RAM.

into the same RAM it loaded from. The Create Clusters component then takes the generated features in P and result of the transitive closure in Q from the same sample set and creates a set of local clusters. The created clusters are stored in B, a set of two RAMs. There are only two of the B RAMs because these clusters are only used for merging. During one iteration, the data from one B RAM is used to merge while the other is being filled with new clusters. Lastly, the Merge Clusters component takes the last created clusters in B and merges them with the global clusters in A. If the hardware is done scanning background data and a new global cluster is formed, the Merge Clusters component raises a flag alerting the user an anomaly has been detected.

An alternative to pipelining would be parallelization. Memory is not an issue in the existing R9 architecture because 90% of the FPGA's memory is available. However, it was discovered that as more random-access memories (RAM) were added, the overhead increased. This is because the input and output signals for the RAMs had to pass through an increased number of multiplexers to get in and out of the RAMs. The overhead grew so large that it was

Algorithm 2 Floyd-Warshall Min-Transitive Closure

Input: Matrix Q, Dimension n

```
for i = 1 to n do
    for j = 1 to n − 1, j ≠ i do
        for k = j + 1 to n do
            q_jk ← max(q_jk, min(q_ji, q_ik))
            q_kj ← q_jk
        end for
    end for
end for
```

not possible to reach the timing goal through parallelization.

The FPGA only has 16% of its space remaining. Also, there are no multipliers available for use with our added hardware. Modifications were made to the algorithm to limit the amount of space used, mainly by limiting the use of divisions and multiplications. When calculating the fuzzy compatibility relation (Equation 1), the difference between sample features is divided by the range of the feature. The division by the range was substituted with a series of logical right shifts. The difference is shifted right to get a result as close as possible to the division with the range. The transformation of this equation was also done for the fuzzy equivalence relation done before merging clusters in Equation 5. This method was tested in software to verify the end result would still be valid.

As described previously, the max-min transitive is calculated by iteratively applying Equation 2 to the fuzzy compatibility relation. This equation is relatively simple and easy to implement in hardware, however it requires $\mathcal{O}(n^3 \log n)$ time where n is the sample size. It was not possible to perform this step in the 8,000 clock cycle limit, so to avoid adding more pipeline stages a different algorithm was used. In [11], the Floyd-Warshall algorithm is shown to be another simple yet quicker method of computing the max-min transitive closure. The algorithm is modified to take advantage of the reflexivity and symmetry of the fuzzy compatibility relation.

The complexity is reduced to $\mathcal{O}(n^3)$ in Floyd-Warshall's method while still only using max and min operations. This method sufficiently speeds up the hardware implementation such that it now can execute in under the 8,000 clock cycle limit.

6. Conclusions

In applications such as the one described, there are instances where things need to be detected that have not been seen before. This paper has demonstrated why the fuzzy clustering algorithm with merging is well-suited for detecting new patterns online. The algorithm used was optimal for applications of this type because of its ability to merge clusters in a way that does not require all data to be re-clustered every iteration. The approach taken to utilize this algorithm in hardware could easily be used for similar applications simply by changing the features.

As seen with the software results, the algorithm was successfully implemented and able to detect a significant surface height difference. Modifications were made to the algorithm to make it easier to design in hardware. Hardware simulations show that the designed hardware outputs the same result as the software when given the same data stream in the required amount of time. Also, when synthesizing the hardware to be put on the FPGA, there is enough remaining space for the hardware and memory needed.

This application can be further expanded upon to produce better results from AFM scans. A user may wish to focus on an object of interest after it has been found. If only the area around the object is scanned, the scan could be taken faster and at a higher resolution. Logic could be added to the hardware described in this paper to automate the process of stopping the current scan when an object is found and set up the new, focused scan.

References

[1] G. Pallotta, M. Vespe, and K. Bryan, "Vessel pattern knowledge discovery from ais data: A framework for anomaly detection and route prediction," *Entropy*, vol. 15, no. 6, pp. 2218–2245, 2013.

[2] W. Wang, T. Guyet, R. Quiniou, M.-O. Cordier, F. Masseglia, and X. Zhang, "Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks," *Knowledge-Based Systems*, vol. 70, pp. 103–117, 2014.

[3] O. Akbilgic and J. A. Howe, "Symbolic pattern recognition for sequential data," *Sequential Analysis*, vol. 36, no. 4, pp. 528–540, 2017.

[4] S. Ghosh and S. K. Dubey, "Comparative analysis of k-means and fuzzy c-means algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 4, 2013.

[5] P. Agarwal, M. A. Alam, and R. Biswas, "Issues, challenges and tools of clustering algorithms," *arXiv preprint arXiv:1110.2610*, 2011.

[6] L. Zhang, J. Lin, and R. Karim, "Sliding window-based fault detection from high-dimensional data streams," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 2, pp. 289–303, 2017.

[7] R. E. Haskell, D. M. Hanna, P. Li, K. Cheok, and G. Hudas, "Finding pattern behavior in temporal data using fuzzy clustering," *Intelligent Enginering Systems Through Artificial Neural Networks*, vol. 10, pp. 703–711, 2000.

[8] A. Kollin, S. Porthun, D. Hanna, C. Otlowski, A. Covyeau, K. LaBelle, M. Lohrer, and J. Gorski, "Design of an efficient object-oriented software for an fpga-based scan probe microscope controller," in *Proceedings of the International Conference on Scientific Computing (CSC)*, p. 178, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[9] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic*, vol. 4. Prentice hall New Jersey, 1995.

[10] S. Xu, S. J. Cruchon-Dupeyrat, J. C. Garno, G.-Y. Liu, G. Kane Jennings, T.-H. Yong, and P. E. Laibinis, "In situ studies of thiol self-assembly on gold from solution using atomic force microscopy," *The Journal of chemical physics*, vol. 108, no. 12, pp. 5002–5012, 1998.

[11] H. Naessens, H. De Meyer, and B. De Baets, "Algorithms for the computation of t-transitive closures," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 4, pp. 541–551, 2002.