Computer Science • 23(3) 2022 https://doi.org/10.7494/csci.2022.23.3.4389

Jakup Fondaj
Zirije Hasani
Samedin Krrabaj

PERFORMANCE MEASUREMENT WITH HIGH-PERFORMANCE COMPUTER USING HW-GA ANOMALY-DETECTION ALGORITHMS FOR STREAMING DATA

Abstract *Anomaly detection for streaming real-time data is very important; more significant is the performance of an algorithm in order to meet real-time requirements. Anomaly detection is very crucial in every sector because, by knowing what is going wrong with data/digital systems, we can make decisions to help in every sector. Dealing with real-time data requires speed; for this reason, the aim of this paper is to measure the performance of our proposed Holt–Winters genetic algorithm (HW-GA) as compared to other anomaly-detection algorithms with a large amount of data as well as to measure how other factors such as visualization and the performance of the testing environment affect the algorithm's performance. The experiments will be done in R with different data sets such as the as real COVID-19 and IoT sensor data that we collected from Smart Agriculture Libelium sensors and e-dnevnik as well as three benchmarks from the Numenta data sets. The real data has no known anomalies, but the anomalies are known in the benchmark data; this was done in order to evaluate how the algorithm works in both situations. The novelty of this paper is that the performance will be tested on three different computers (in which one is a high-performance computer); also, a large amount of data will be used for our testing, as will how the visualization phase affects the algorithm's performance.*

Keywords time-series data, HW-GA, anomaly detection, big streaming data, Numenta, COVID-19 data set, high-performance computer, Libelium sensor data, e-dnevnik

Citation Computer Science 23(3) 2022: 395–410

1. Introduction

Performance measurement is one of the main mechanisms for testing one algorithm for its speed; this is crucial when dealing with real-time big data. Anomaly detection in real time requires speed to detect anomalies; to deal with this, we test the algorithm on different computers with different performance levels to improve the speed of HW-GA. In research that was completed earlier [7], we proposed HW-GA for detecting anomalies with large amount of real-time data. We tested the accuracy of the algorithm by comparing it to several other algorithms that we singled out from previous research (such as ARIMA [9], moving average [1], and Holt-Winters [3]). Also, the performance of this algorithm is tested in this research [6]; however, there is no research that measures the performance of HW-GA with a high-performance computer in order to see if it affects the execution time and CPU usage. In addition, we did not test how the visualization process of the algorithm affects the execution time, and we did not test the algorithm with a large amount of data (more than 270,000 records) for its performance and accuracy. The contributions of this paper are numerous:

1) It shows how the performance of the algorithm is affected by the computer's performance on which the algorithm is tested.
2) It compares the algorithm with other algorithms.
3) It tests with a larger amount of data to see whether the amount of data that is used for testing affects the algorithm's performance.
4) It tests how a visualization of the results of this algorithm affects the execution times of the other algorithms.

In addition to our previous work [7], we tested the proposed algorithm against other selected algorithms with high-performance computers in this paper in order to see whether the performance of the algorithm was affected by the performance of the devices where the algorithms are tested.

When dealing with large amounts of data, it should be kept in mind that this is characterized by three basic characteristics: the volume, veracity, and variety of the data (hence, the need for performance testing in order to meet the speed characteristics of large amounts of data). This is a vast field of research, as it involves algorithms from different disciplines. First, it is important to specify the data to be analyzed in order to know how we make the algorithm selection.

This research used qualitative and quantitative research methods. Qualitative research methods were used in this research for an existing literature review, where theories and concepts that are related to the performance measurement of anomaly-detection algorithms in big data were reviewed. On the other hand, a quantitative research method was used for the experiments that will be conducted in order to measure the evaluation time and CPU usage of HW-GA as compared to the other algorithms. The research was carried out by using several specific methodological procedures and research techniques: analysis, classification, comparison, synthesis, induction, deduction, and experimentation.

First of all, the characteristics of the massive data streams that are created as a result of events in complex computer systems and business transactions were analyzed. The study then analyzed the corresponding existing methodologies and algorithms for detecting anomalies over such a stream type, whereby the anomalies were expected to be of a contextual and collective type. The methodologies for measuring the performance of the anomaly-detection algorithms that we used are the execution time of the algorithm and the CPU usage during the phase when the algorithm was running. Comparison methods were used because the testing will be done for different amounts of data in order to see whether the amount of data affected the performance of the algorithms. In addition, three computers with different levels of performance were used to do the experiments in order to compare whether a high-performance computer reduced the execution times and CPU usage of the algorithms. Also, another analysis that we performed was to test whether HW-GA's execution time was affected from the last part of the algorithm's execution (which was a visualization of the results and how much it increased the execution time).

The induction and deduction method was used to lock in the possibilities of using HW-GA for detecting anomalies in real time in large amounts of data or more-specific massive data streams. Free software environment R[1] was used for doing the experiments as well as the statistical analyses and visualizations of the results. Benchmarks and real-time data were used to test the algorithm, as were the Numenta benchmark [10] database, real-time data from the e-dnevnik application[2] (which is an electronic education system in North Macedonia), Libelium sensor data for air-quality measurement, and the COVID-19 data set [11].

The remaining chapters of this paper are as follows. The second chapter shows related works, then the third chapter presents the benchmark and real data that was used for the experiments. A brief introduction to the algorithms used for these experiments is given in the fourth part, then the fifth section describes the comparison of performance of the algorithms. Finally, the sixth chapter discusses the results and conclusions from this research.

2. Related work

In research that was completed earlier [7], we proposed HW-GA for detecting anomalies in large amounts of real-time data. We tested the accuracy of the algorithm there, comparing it to several other algorithms that we singled out from previous research (such as ARIMA [9], moving average [1], and Holt-Winters [3]). Also, the performance of these algorithms was tested in this research [6]; however, there has been no research that measures the performance of HW-GA with a high-performance computer in order to see whether it affects the execution time and CPU usage. The experiments were done with real data from e-dnevnik, IoT sensor data (which we collected from Smart Agriculture Libelium sensors), COVID-19 [12], and the NAB benchmark [10].

[1]R Version 3.4.3. URL: https://www.r-project.org
[2]http://ednevnik.edu.mk

A comparison of anomaly-detection algorithms for real-time big data was done here [5], where we evaluated many algorithms (DBSCAN, MAD, moving average control chart, ARIMA, Twitter, etc.) in order to select the fastest methods. So, the most important aspects that we considered in order to find an anomaly-detection algorithm that was suitable for future implementation in an online environment was the execution time (complexity), the CPU usage, and the satisfactory quality of the algorithm (measured through the TP-True Positive, FP-False Positive, FN-False Negative, and TN-True Negative anomalies that were found).

The best algorithms that were selected from this research (ARIMA and moving average) were compared with HW-GA [7] and Holt-Winters, where we tested the correctness of HW-GA in our previous research [7]. In this paper, we tested the performance/speed and CPU usage of HW-GA and compared it with Holt-Winters, ARIMA, and moving average in this research. The authors of [10] proposed a benchmark Numenta anomaly benchmark (NAB); this benchmark was used in our research. The authors of [14] proposed an online and unsupervised anomaly-detection algorithm for streaming data using an array of sliding windows and probability density-based descriptors (PDDs) based on these windows. The experimental results and performances were presented based on the Numenta anomaly benchmark. Martin et al. [2] showed how to find anomalies in a video stream by using the Markov model. The time intervals of the data that were used for the experiments were 1 hour, 1 day, and 3 days. The anomaly-detection method proposed by these authors controlled all of the new streaming data that came in real time for possible anomalies.

Filipe Falcão et al. [4] experimentally evaluated a pool of 12 unsupervised anomaly-detection algorithms on five data set attacks. Their results allowed them to elaborate on a wide range of arguments, from the behavior of an individual algorithm to the suitability of a data set for anomaly detection. They identified the families of algorithms that were more effective for intrusion detection as well as the families that were more robust to the choice of configuration parameters. A proposal of an anomaly-detection framework for video patches was proposed in [15]. The method they proposed is named HMOF and is characterized with low computation time and high efficiency. They experimentally showed that their proposed method worked better than other methods for anomaly detection in real time.

Other research in this field [13] proposed a data-stream anomaly-detection algorithm that was mainly based on a self-set threshold with extreme value theory (ESOD). The threshold in this proposed algorithm was updated in real time in order to be adopted for real-time streams; the algorithm showed good usability and high efficiency. One algorithm for anomaly detection in manufacturing equipment was proposed in [8]. In their method, they implemented it in Java and C++; brief technical details about how their method worked were given here.

Another algorithm for anomaly detection in both real-time and batch data was proposed in [12]. The authors used K-Means clustering for anomaly detection. The aim of this algorithm was to detect any potential problems of offshore rotating machinery.

They compared their method with the signal analysis method. Having this view on the actual research, we can see that research has not been made that measures the performance of HW-GA with different computers of differing performance levels and also tests the algorithm with larger data sets while analyzing how the visualization process affects its performance.

3. Benchmark data sets and testing environments

Different databases with different sizes were used for our experiments in order to test the performance of algorithms using computers with different performance levels (testing environments). The experiments were performed with three computers[3] with different performance levels as follows:

1) Lenovo Legion Y520 – Intel core i7-7700HQ and 16 GB RAM.
2) MSI GL 75 Leopard – Intel (R) Core (TM) i7-10750H CPU @ 2.60 GHz, 2592 Mhz, 6 Cores(s), 12 Logical Processor(s)
3) High-performance computer – Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz (two processors) with 192 GB RAM (191 GB usable).

Real data from e-dnevnik, the IoT sensors (from the Smart Agriculture Libelium sensors), the COVID-19 data set [12], and the NAB benchmark [10] was used in our experiments. NAB contains data sets with real-world labeled data files from across multiple domains as well as the associated anomaly detectors that are applicable for the streaming data. We used three NAB data sets: HotGym (the energy consumption from one gym center in Australia), CPU utilization, and NycTaxi (the number of rides for NYC taxis). We also used the COVID-19 data set (this data set had everyday-level data on the number of influenced cases, recoveries, and deaths from January 2020 to December 2020 [5] for different countries) and Libelium sensor data (containing air-quality sensor data such as CO2, NOISE, etc.); this data was time-series data, so the number of cases on any given day was the cumulative number. The data was available from Jan 22, 2020, through Dec 6, 2020. The data set had eight columns (172,480 records – rows), as did our e-dnevnik. Parts of the data sets are shown in Table 1 below. The data sets contained two main attributes (timestamps and values), which were generated in a log file. The first three data sets contained benchmark data with known anomalies, and the last three from e-dnevnik, the Libelium sensors, and the COVID-19 data set contained real data where the anomalies are unknown. We tested HW-GA with all of them in order to see whether the algorithm could find the anomalies; even when we did not specify the intervals of the anomalies, we followed the logic of the training and test sets. The first two data sets contained real values, and the last three had integer values. The following figure (Fig. 1) presents the COVID-19 data (the anomaly in this data is indicated with red circle).

[3]The equipment that was used for the experiments in this research was financed by "The development and implementation of a PhD Program in ICT for the Kosovo Education System (PhDICTKES) Project No: 609990-EPP-1-2019-1-SE-EPPKA2-CBHE-JP"

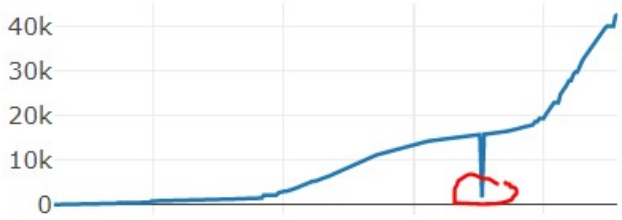

Figure 1. Anomalies in COVID-19 data

Table 1
Part of benchmark data used for experiments

HotGym		CPU utilization		Nyctaxi		
Timestamp	KW energy consumption	Timestamp	Metric Values	Timestamp	Value	
7/2/2010 0:00	21.2	4/10/2014 0:04	93.1456	7/1/2014 0:00	10,844	
7/2/2010 1:00	16.4	4/10/2014 0:24	94.5935	7/1/2014 0:30	8127	
7/2/2010 2:00	4.7	4/10/2014 0:44	93.521	7/1/2014 1:00	6210	
Rtime e-dnevnik		COVID-19		Libelium SmartCityPro		
Timestamp	KW energy consumption	Timestamp	Metric Values	Sensor	Value	Timestamp
6/13/2016 0:00	6431	1/22/2020	1	CO2	–1	6/8/2021 14:55
6/13/2016 0:00	345	1/22/2020	14	PRES	96821.63	6/8/2021 14:55
6/13/2016 0:00	354	1/22/2020	6	HUM	41.6	6/8/2021 14:55

This data set had everyday-level data on the number of influenced cases, recoveries, and deaths from January 2020 to December 2020 (for different countries) [12]. The data is everyday data, and the number of positive cases grows (which means that, if the number of positive cases falls very quickly for one day, this means something has gone wrong based on the previous trend of the positive case growth).

4. Algorithms used for testing

Our proposed Holt-Winters genetic algorithm (HW-GA) [7] is an anomaly-detection algorithm for streaming data. HW-GA was compared to ARIMA [9] and the moving average algorithm [1] (which is used mostly for modeling univariate time series). The comparison was done also with the existing Holt-Winters algorithm and some

version of a modified Holt-Winters. In Figure 2, the model for the HW-GA method is presented (which is composed of several steps). Starting from the input step where the data set came, the annotated anomaly interval is defined as the interval where one anomaly may occur. Then, the next stage is the computation of the anomaly-detection parameters by using genetic algorithm. As a result, we obtain optimal values for the algorithm parameters that are used for the algorithm evaluation and anomaly detection [7]. The main stage is the next stage, where the detection of anomalies is done for data streams that arrive in real time. This stage used the optimal parameters that were computed from the second stage [7]. The fourth stage is the results that are checked by a human and classified as TP, FP, or FN. The result taken in this step is used again in the second stage for the further optimization of the anomaly-detection parameters [12].

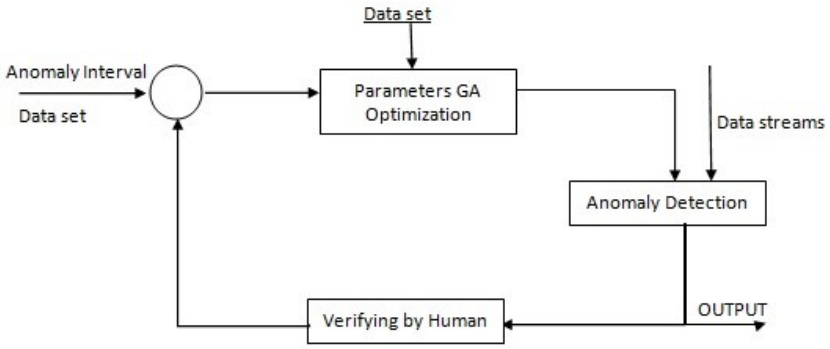

Figure 2. Model for HW-GA method for anomaly detection [7]

5. Performance measurement of HW-GA anomaly-detection algorithm with high-performance computer

The performance of HW-GA [7] is tested in this paper, and the algorithm is compared with five other algorithms. The execution time and CPU usage were measured on three different computers. The testing was done in two forms. The first three tables show the results for the algorithms with a visualization of the results, and the last three tables show the results of the algorithms without the visualization process. Also, a comparison of the results is shown by the charts in Figures 3–6.

The algorithmś evaluation focus was on execution response time and CPU usage. These two parameters were measured in the running times of the algorithms. The algorithms were implemented in the R language; the beginnings and ends of the codes have a time measurement code that shows us the time that each algorithm takes to be executed. On the other side, the CPU usage is monitored in real time when the algorithm is running.

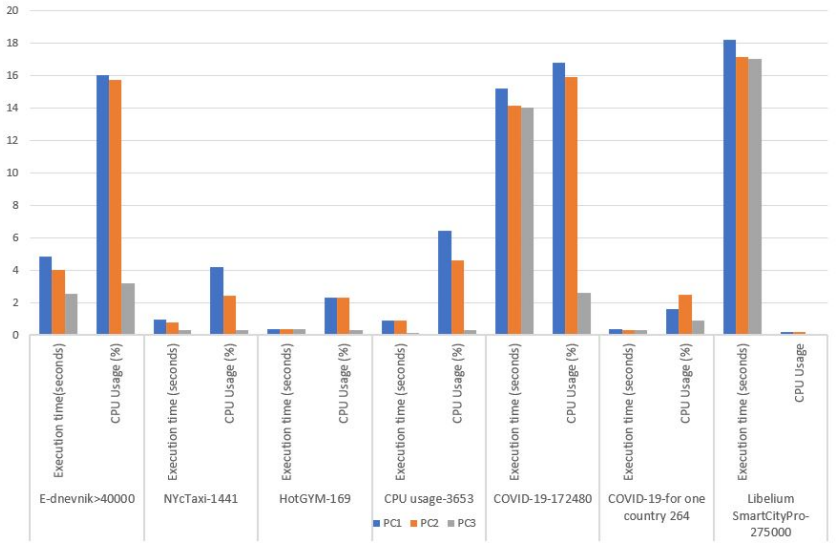

Figure 3. Comparison of performance for three computers for HW-GA with visualization process

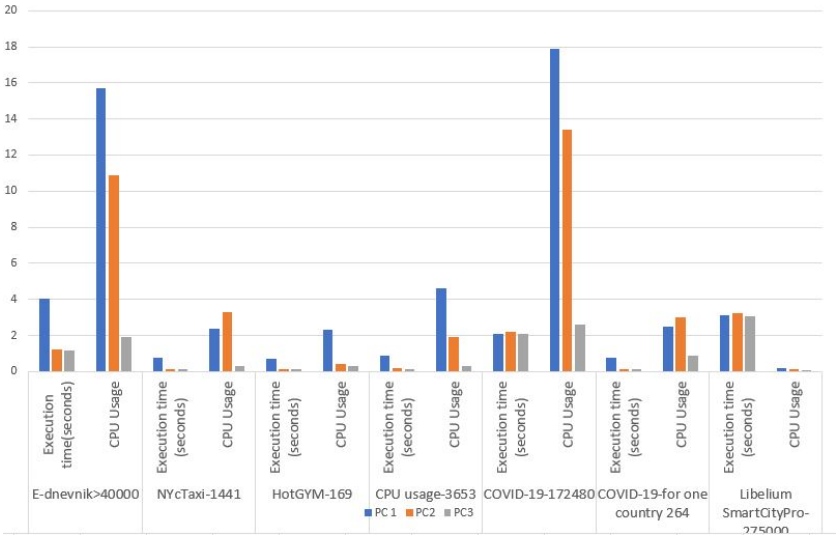

Figure 4. Comparison of performance for three computers for HW-GA with visualization process

In Figures 5 and 6, we visually show a comparison between the performance of HW-GA with the visualization process and without visualization. We can see that the visualization affects the performance; for example, the maximum value for the execution time was 15 seconds with visualization, but the maximum value was 4 seconds without the visualization. The same is not true for CPU usage, however; here,

we conclude that, if the amount of data is large, the CPU usage is almost the same with or without the visualization process.

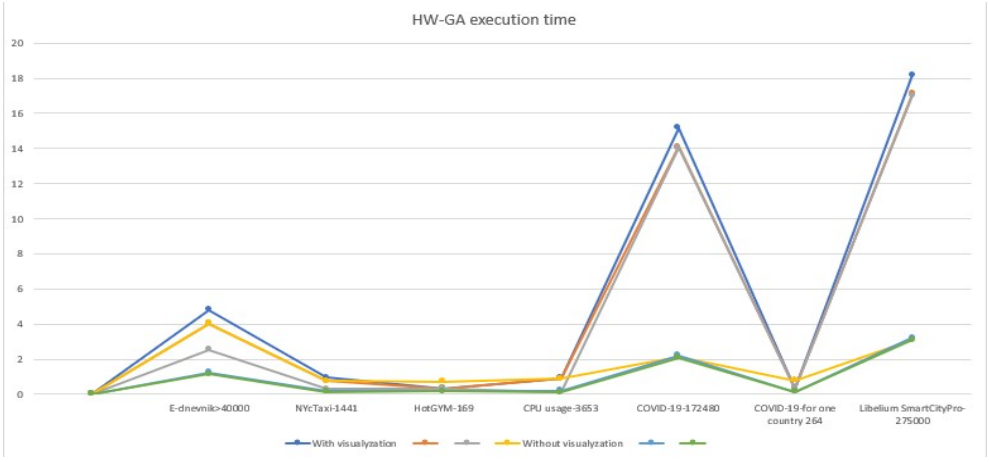

Figure 5. Comparison of execution times for three computers for HW-GA

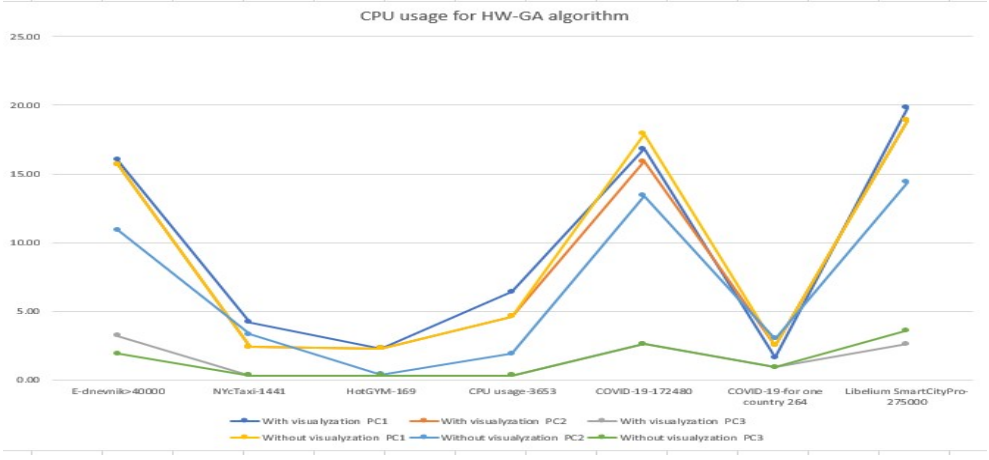

Figure 6. Comparison of CPU usage for three computers for HW-GA

These parameters are important for us because HW-GA will need to work in a real-time environment. Tables 2–7 show the execution times and CPU usage for six algorithms (which are each compared with HW-GA). The experiments were done on real data and benchmark data as described above. What we can see from these tables is that the execution times and CPU usage were smaller when the performance of the computer was higher; this means that a higher-performance computer results in smaller execution time and CPU usage.

Table 2

Experimental results from CPU usage and execution time with computer 1 with visualization process

	E-dnevnik > 40,000		NYcTaxi-1441		HotGYM-169		CPU usage-3653		COVID-19 – 172,480		COVID-19 – for one country 264		Libelium SmartCityPro-275,000	
Algorithms	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]
HW GA	4.80729	16.0	0.9400001	4.2	0.3287289	2.3	0.9073496	6.4	15.1705792	16.8	0.330117	1.6	18.1705792	19.8
HW calc. MASE	5.578303	17.1	0.34126	4.6	0.3899832	5.3	2.389023	15.7	20.354	18.6	0.413404	6.3	23.354	21.6
HW def. MASE(k)	5.281588	16.5	0.3370709	4.5	0.3529601	4.0	2.45762	15.0	21.13686	17.3	0.44556	5.6	24.13686	20.3
HW def. MASE(k,n)	5.249209	17.2	0.3370981	4.3	0.3782699	2.5	2.391995	15.5	20.58003	17.3	0.4036062	6.4	24.58003	20.3
ARIMA	0.1375349	5.0	0.3897381	1.0	0.346854	1.2	1.942476	8.7	17.47428	17.4	4.942397	16.9	21.47428	20.4
MA	40.51032	32.1	0.5697041	2.9	0.2228181	1.2	6.34102	16.8	24.39901	17.5	0.316108	1.2	28.39901	20.5

Table 3

Experimental results from CPU usage and execution time with computer 2 visualization process

	E-dnevnik > 40,000		NYcTaxi-1441		HotGYM-169		CPU usage-3653		COVID-19 – 172,480		COVID-19 – for one country 264		Libelium SmartCityPro-275,000	
Algorithms	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]
HW GA	4.026123	15.7	0.7509849	2.4	0.3287209	2.3	0.9024837	4.6	14.1093571	15.9	0.294966	2.5	17.1093571	18.9
HW calc. MASE	4.339186	15.5	0.851656	2.5	0.806777	2.7	2.462865	9.2	15.04244	16.3	0.839512	2.6	18.04244	19.3
HW def. MASE(k)	4.599722	15.7	0.7875409	2.7	0.803364	2.6	2.420955	9.1	15.37323	16.1	0.806761	2.7	18.37323	19.1
HW def. MASE(k,n)	4.392256	15.8	0.780865	2.6	0.785208	2.1	2.221919	7.8	15.4715	16.2	0.8290341	2.9	18.4715	19.2
ARIMA	3.183986	16.1	0.204	0.1	0.1250021	0.9	1.171567	4.7	10.67319	16.2	2.405716	13.0	13.67319	19.2
MA	14.43228	16.1	0.3436749	2.0	0.1249652	1.0	3.9545519	16.4	14.89439	16.2	0.140554	0.5	17.89439	19.2

Table 4

Experimental results from CPU usage and execution time with high-performance computer 3 visualization process

	E-dnevnik > 40,000		NYcTaxi-1441		HotGYM-169		CPU usage-3653		COVID-19 – 172,480		COVID-19 – for one country 264		Libelium SmartCityPro-275,000	
Algorithms	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]
HW GA	2.544	3.2	0.2908518	0.3	0.326585	0.3	0.1093949	0.3	14.023	2.6	0.295428	0.9	17.023	2.6
HW calc. MASE	2.681	3.0	0.3747468	0.3	0.3927391	0.8	2.697717	2.7	22.795	3.6	0.392076	0.9	25.795	5.6
HW def. MASE(k)	2.788704	3.1	0.3504999	0.1	0.3390379	0.9	2.83155	3.0	25.21961	3.7	0.401022	1.0	28.21961	5.7
HW def. MASE(k,n)	2.696658	3.2	0.362524	0.3	0.3523519	0.6	2.85051	3.1	24.96	3.5	0.3547509	1.0	27.96	5.5
ARIMA	2.301431	2.3	0.4373889	0.5	0.3363271	0.3	2.388546	2.5	18.35616	3.7	0.3436611	0.1	21.35616	5.7
MA	6.133697	3.5	0.749815	0.6	0.3050261	0.1	7.019266	3.4	1 min	3.7	0.359283	0.1	1.1 min	5.7

Table 5

Experimental results from CPU usage and execution time with computer 1 (lenovo) without visualization process

	E-dnevnik > 40,000		NYcTaxi-1441		HotGYM-169		CPU usage-3653		COVID-19 – 172,480		COVID-19 – for one country 264		Libelium SmartCityPro-275,000	
Algorithms	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]
HW GA	4.026123	15.7	0.7509849	2.4	0.710241	2.3	0.9024837	4.6	2.1093571	17.9	0.774966	2.5	3.1093571	18.9
HW calc. MASE	4.339186	15.5	0.851656	2.5	0.806777	2.7	2.462865	9.2	2.559318	17.3	0.839512	2.6	3.559318	18.3
HW def. MASE(k)	4.599722	15.7	0.7875409	2.7	0.803364	2.6	2.420955	9.1	2.520335	17.1	0.806761	2.7	3.520335	18.1
HW def. MASE(k,n)	4.392256	15.8	0.780865	2.6	0.785208	2.1	2.221919	7.8	2.52127	17.7	0.8290341	2.9	3.52127	18.7
ARIMA	3.183986	16.1	0.204	0.1	0.1250021	0.9	1.171567	4.7	10.67319	16.2	2.405716	13.0	11.67319	17.2
MA	14.43228	16.1	0.3436749	2.0	0.1249652	1.0	3.9545519	16.4	7.932833	17.0	0.140554	0.5	8.932833	18.0

Table 6
Experimental results without result visualization from CPU usage and execution time with computer 2 without visualization process

	E-dnevnik > 40,000		NYcTaxi-1441		HotGYM-169		CPU usage-3653		COVID-19 – 172,480		COVID-19 – for one country 264		Libelium SmartCityPro-275,000	
Algorithms	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]
HW GA	1.245121	10.9	0.1595991	3.3	0.163022	0.4	0.19181	1.9	2.212404	13.4	0.1569829	3.0	3.212404	14.4
HW calc. MASE	0.7110801	12.8	0.1754942	4.2	0.1775041	0.9	0.370883	2.7	2.590235	17.4	0.1874759	.1	3.590235	18.4
HW def. MASE(k)	0.8305309	10.7	0.174639	4.0	0.173562	0.5	0.3893731	4.0	2.729638	17.1	0.1726429	3.4	3.729638	18.1
HW def. MASE(k,n)	0.6914442	9.5	0.179064	2.9	0.1916461	2.0	0.37221	4.2	2.722628	17.1	0.1818118	4.2	3.722628	18.1
ARIMA	0.1375349	5.0	0.3897381	1.0	0.346854	1.2	1.942476	8.7	17.47428	17.4	4.942397	16.9	18.47428	18.4
MA	40.51032	32.1	0.5697041	2.9	0.2228181	1.2	6.34102	16.8	6.39901	17.5	0.316108	1.2	7.39901	18.5

Table 7
With high-performance computer 3 without visualization process

	E-dnevnik > 40,000		NYcTaxi-1441		HotGYM-169		CPU usage-3653		COVID-19 – 172,480		COVID-19 – for one country 264		Libelium SmartCityPro-275,000	
Algorithms	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]	Execution time [s]	CPU usage [%]
HW GA	1.155829	1.9	0.1718299	0.3	0.1608318	0.3	0.1093949	0.3	2.0876789	2.6	0.1474619	0.9	3.0876789	3.6
HW calc. MASE	1.9673611	2.3	0.180125	0.3	0.2186959	0.8	0.4934959	0.9	3.077389	3.6	0.2186918	0.9	4.077389	4.6
HW def. MASE(k)	1.165854	2.6	0.2030749	0.1	0.187453	0.9	0.468631	0.8	3.680816	3.5	0.2030659	1.0	4.680816	4.5
HW def. MASE(k,n)	1.046613	2.1	0.2030718	0.3	0.2099216	0.6	0.547925	1.1	3.581017	3.5	0.2030721	1.0	4.581017	4.5
ARIMA	2.301431	2.3	0.4373889	0.5	0.3363271	0.3	2.388546	2.5	18.35616	3.7	0.3436611	0.1	19.35616	4.7
MA	6.133697	3.5	0.749815	0.6	0.3050261	0.1	7.019266	3.4	7.5	3.7	0.359283	0.1	8.5	4.7

6. Result discussion

Measuring the performance of algorithms is not an easy task; however, by doing this, we can come up with many conclusions and comparisons. For these experiments, we used three computers with different performance levels where the CPU usage and execution time are measured for each algorithm. We analyzed how the visualization phase of an algorithm affects its performance and how the amount of data also affects its performance by using different-sized data sets.

From our results, we can see that the execution time was 2.544 seconds faster in HW-GA on the third computer using e-dnevnik data with more than 40,000 data bits; it was 17 seconds faster with a larger data set of more than 270,000 items. Also, the CPU usage was smaller when using real data. The execution time also depends on the amount of data to be tested; this also affects CPU usage. However, when this was compared to HW-GA, it showed better results.

With the real data, the last algorithm (MA) showed greater CPU usage 20.5% for the first computer; in the third computer, this value was more than 50% smaller 5.7%. This was not the same situation with the benchmark data; this happened because the amount of data from the IoT sensor that we collected from the Smart Agriculture Libelium Sensors' real data was greater than 270,000, whereas this amount was much smaller in the others (NYcTaxi – 1441 records; HotGYM – 169 records; CPU usage – 3653 records).

The testing showed that the execution time and CPU usage was more than 50% greater with the visualization results; this means that it affected the algorithm's execution time. For this reason, we concluded that, if we need to find just one anomaly, we can do so without visualization. If we need to analyze where an anomaly is in more detail, we need to visualize the results. This means that we may know when to use visualization and where/when not to. For example, we may automatically check whether one anomaly is found – if so, a visualization should be be performed; if not, then it is not necessary.

Also, we found that the performance of the computer affected the CPU usage as well as the execution time, which means that a computer with greater performance abilities results in lower CPU usage and execution time. The results and comparisons from the experiments are visualized in Figures 3–6. What differs is the algorithm execution time, which increases when the amount of data is large; however, what is important is that a high-performance computer reduces the execution time of the algorithm. These results allow us to conclude that we can improve HW-GA's performance by increasing the computer's performance. We can conclude that we need to modify HW-GA by adding more parameters in the process of finding optimal parameters in order to see whether it affects the algorithm's execution time. Based on these facts that are described here, we can say that our algorithm outperforms the others algorithm in both measurement parameters (execution time and CPU usage); also, better results are shown with a large amount of data, which is very important for big data.

7. Conclusion

Measuring the performance of anomaly detection in a real-time big-data algorithm is very important because it needs to be fast to deal with real-time data. The aims of this paper were many.

First of all, the performance of the algorithm was tested on three different computers with different performance levels to analyze how it affected the performance of HW-GA. Then, the amount of data that was used for the testing was larger for more than 270,000 records to see how the algorithm's performance would be affected by increasing the amount of data. Third, we analyzed how the visualization process in this algorithm affected the algorithm's execution time.

Based on this research we can conclude that HW-GA is efficient in regards to the algorithm's execution time and CPU usage. Also, the performance of the computer affects positively the performance of the algorithm by reducing the execution time and CPU usage. HW-GA shows better results based on the measurement parameters (execution time and CPU usage) with a large amount of data, which is very important for big data. In all of our testing environments, the algorithm showed less CPU usage and lower execution times when compared to the other algorithms. These results are shown in the tables above.

We also conclude that the visualization of HW-GA's results affects more than 50% of its execution time. For this reason, we recommend that visualization should be performed only when one anomaly is found in order to better analyze where it happens. In other cases where one anomaly is not found, visualization should not be performed. The proposed method is meant for time-series log data and has not yet been tested on other types of data to see whether the results would be the same. This is a limitation of our method and will be tested in the future; based on the results, we will adapt it for other types of data.

In our continuous work, we will modify the algorithm by adding more parameters in the optimization process with a genetic algorithm in order to see whether it affects the algorithm's performance.

References

[1] Contributed packages, CRAN. https://cran.r-project.org/web/packages/.

[2] Boldt M., Borg A., Ickin S., Gustafsson J.: Anomaly detection of event sequences using multiple temporal resolutions and Markov chains, *Knowledge and Information Systems*, vol. 62, 2020. doi: 10.1007/s10115-019-01365-y.

[3] Ekberg J., Ylinen J., Loula P.: Network behaviour anomaly detection using Holt-Winters algorithm. In: *2011 International Conference for Internet Technology and Secured Transactions*, pp. 627–631, 2011.

[4] Falcão F., Zoppi T., Barbosa Vieira da Silva C., Santos A., Fonseca B., Ceccarelli A., Bondavalli A.: Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection. In: *SAC '19: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 318–327, 2019. doi: 10.1145/3297280.3297314.

[5] Hasani Z.: Robust anomaly detection algorithms for real-time big data: Comparison of algorithms. In: *2017 6th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–6, 2017. doi: 10.1109/MECO.2017.7977130.

[6] Hasani Z.: Anomaly Detection Algorithms for Streaming Data: Performance Comparison, *Journal of Computer Science*, vol. 16(7), pp. 950–955, 2020. doi: 10.3844/jcssp.2020.950.955.

[7] Hasani Z., Jakimovski B., Velinov G., Kon-Popovska M.: *An Adaptive Anomaly Detection Algorithm for Periodic Data Streams: 19th International Conference, Madrid, Spain, November 21–23, 2018, Proceedings, Part I*, pp. 385–397, 2018. doi: 10.1007/978-3-030-03493-1˙41.

[8] Jankov D., Sikdar S., Mukherjee R., Teymourian K., Jermaine C.: Real-Time High Performance Anomaly Detection over Data Streams: Grand Challenge. In: *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, p. 292–297, DEBS '17, Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3093742.3095102.

[9] Kasunic M., McCurley J., Goldenson D., Zubrow D.: *An Investigation of Techniques for Detecting Data Anomalies in Earned Value Management Data Software Engineering Measurement and Analysis (SEMA)*, 2011.

[10] Lavin A., Ahmad S.: Evaluating Real-Time Anomaly Detection Algorithms – The Numenta Anomaly Benchmark. In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 38–44, 2015. doi: 10.1109/ICMLA.2015.141.

[11] Srk: Novel Corona virus 2019 dataset, 2021. https://www.kaggle.com/datasets/sudalairajkumar/novel-corona-virus-2019-dataset.

[12] Wang Z., Zhou Y., Li G.: Anomaly detection for machinery by using Big Data Real-Time processing and clustering technique. In: *Proceedings of the 2019 3rd International Conference on Big Data Research*, p. 30–36, ICBDR 2019, Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3372454.3372480.

[13] Yuanyan L., Xuehui D., Yi S.: Data streams anomaly detection algorithm based on self-set threshold. In: *Proceedings of the 4th International Conference on Communication and Information Processing*, pp. 18–26, ICCIP '18, Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3290420.3290451.

[14] Zhang L., Zhao J., Li W.: Online and Unsupervised Anomaly Detection for Streaming Data Using an Array of Sliding Windows and PDDs, *IEEE Transactions on Cybernetics*, vol. 51(4), pp. 2284–2289, 2019. doi: 10.1109/TCYB.2019.2935066.

[15] Zhu H., Liu B., Lu Y., Li W., Yu N.: Real-time Anomaly Detection with HMOF Feature. In: *ICVIP 2018: Proceedings of the 2018 the 2nd International Conference on Video and Image Processing*, pp. 49–54, 2018. doi: 10.1145/3301506.3301510.

Affiliations

Jakup Fondaj
jf13459@seeu.ecu.mk

Zirije Hasani
zirije.hasani@uni-prizren.com

Samedin Krrabaj
Wsamedin.krrabaj@uni-prizren.com

Received: 13.07.2021
Revised: 09.11.2021
Accepted: 13.11.2021