I.J. Modern Education and Computer Science, **2014, 6, 58-70**
Published Online June 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2014.06.08

A Harmony Search Algorithm with Multi-pitch Adjustment Rate for Symbolic Time Series Data Representation

Almahdi M. Ahmed[1, 2.]
[1]University Kebangsaan Malaysia /Center for Artificial Intelligence Technology,
Bangi Selangor, 43600, Malaysia
[2]Sebha University /Faculty of Education / Department of computer science, Traqan, 18785, Libya
Email: sheriftsm@gmail.com

Azuraliza Abu Bakar[1] **and Abdul Razak Hamdan**[1]
[1]University Kebangsaan Malaysia /Center for Artificial Intelligence Technology,
Bangi Selangor, 43600, Malaysia
Email: {aab, arh }@ftsm.ukm.my

***Abstract*—**The representation task in time series data mining has been a critical issue because the direct manipulation of continuous, high-dimensional data is extremely difficult to complete efficiently. One time series representation approach is a symbolic representation called the Symbolic Aggregate Approximation (SAX). The main function of SAX is to find the appropriate numbers of alphabet symbols and word size that represent the time series. The aim is to achieve the largest alphabet size and maximum word length with the minimum error rate. The purpose of this study is to propose an integrated approach for a symbolic time series data representation that attempts to improve SAX by improving alphabet and word size. The Relative Frequency (RF) binning method is employed to obtain alphabet size and is integrated with the proposed Multi-pitch Harmony Search (HS^{MPAR}) algorithm to calculate the optimum alphabet and word size. RF is used because of its ability to obtain a sufficient number of intervals with a low error rate compared to other related techniques. HS^{MPAR} algorithm is an optimization algorithm that randomly generates solutions for alphabet and word sizes and selects the best solutions. HS algorithms are compatible with multi-pitch adjustment. The integration of the RF and HS^{MPAR} algorithms is developed to maximize information rather than to improve the error rate. The algorithms are tested on 20 standard time series datasets and are compared with the meta-heuristic algorithms GENEBLA and the original SAX algorithm. The experimental results show that the proposed method generates larger alphabet and word sizes and achieves a lower error rate than the compared methods. With larger alphabet and word sizes, the proposed method is capable of preserving important information.

***Index Terms*—**Data Mining, Date Representation, Time Series, Discretization, Harmony Search Algorithm.

I. Introduction

Time series data are produced, continued, and processed within a wide range of application domains in various fields such as economics, engineering, science, medicine and sociology. A time series is a sequence of observed real-values, usually stamped with time by [1, 2]. In the context of time series data mining, the fundamental problem relates to ways of represent time series data. Several other important concepts that must be considered when using time series include pre-processing, time series detection, similarity computation and mining operations.

The two main characteristics of a time series are the number of segments (word size) and the number of values (alphabet size) required to represent continuous values in the series. The alphabet size partitions the range of possible time series values into a set of regions. Each region corresponds to a specific symbol, and each measurement value is thus uniquely mapped into the symbol of the region in which it falls. The number of regions (symbols) reflects the word size and level of resolution for the information that is retained. These types of data require huge amounts of data storage. Therefore, it is desirable to reduce word size and alphabet size to maintain the most important features of the series (information loss), information loss is produced after time series reduction based on alphabet and word size. Many approaches for working with time series data are focused primarily on data representation [3-5]. Most of the representation algorithms require, as an input, the parameters alphabet size and word size. However, it may be very difficult to know the best values for alphabet and word size in advance in real-world applications. Hence, word size and alphabet size need to be analyzed extensively in order to find the optimal size of the given time series data set [6-8]. In [5] have proposed the SAX

approach to overcome the problems created by time series representations. However, the SAX algorithm does not clearly show how to define word and alphabet size using a time series data set. There are other various algorithms [6, 9] that attempt to solve the problem of defining the optimal (minimum) word and alphabet size, but these methods do not result in minimal information loss. Information loss leads to the destruction of important details contained in the original time series data. These considerations have yet to be discussed in proofs of proposed algorithms; often, researchers attempt only to compress datasets and improve the classification accuracy, which could lead to a loss of some important sensitive information, for example in weather and financial applications, in which the details of the information in the data should not be removed. Such a removal could result in essential information and hidden patterns becoming destroyed. The main objective of this study is to propose an improvement Harmony Search algorithm for symbolic time series data representation. The improvement algorithm takes in account and finds the optimum (maximum) word and alphabet size, thus reducing the loss of information [10].

This paper is organized into five sections. The next section will discuss the background and related work that has been performed on comparison techniques. Section 3 introduces the concept of the proposed methods: RF, HS and SAX^{++} algorithms. The experimental design, results, and discussion are reviewed in Section 4, and Section 5 will conclude our study

II. Related Work

Most data mining methods for time series data are assumed to apply to discrete time series [2]. However, in most applications, these methods generate and use floating point data types. Therefore, there are many approaches to represent time series data in which the data are floating point values. Many approaches and techniques that focus on time series data representation problems have been proposed over the past decade. The first technique that was suggested for time series data representation is the discrete-time Fourier transform (DFT) [3]. The DFT is used to transform a sequence from the time domain to a point in the frequency domain. Choosing the k first frequencies and then representing each sequence as a point in the k-dimensional space achieves this goal. Another technique is the Wavelet Transform (WT) or the Discrete Wavelet Transform (DWT) [4]. The DWT has been found to be effective in replacing the DFT for many applications. The fundamental idea behind wavelets is to analyze the data according to scale. Indeed, some researchers in the wavelet field believe that, with wavelets, one is adopting a whole new mind set or perspective for data processing.

Numerosity reduction technique for time series data reduction was improved by [11] the proposed numerosity reduction, to speed up one-nearest neighbor DTW. While the idea of numerosity reduction for nearest-neighbor classifiers has a long history, it show here that it can leverage off an original observation about the relationship between dataset size and DTW constraints to produce an extremely compact dataset with little or no loss in accuracy. The ideas were tested with a comprehensive set of experiments, and show that it can efficiently produce extremely fast accurate classifiers.

One representation scheme that uses a symbolic representation of the data is called the SAX. SAX represents the time series in terms of word size (w) and alphabet size (a). It thus reduces the dimensionality of the time series into a number of symbolic representations [5]. Their representation is unique in that it reduces the size of the set while also allowing distance measures to define the symbolic approach; smaller distance measures correspond to periods that are defined in the original series. This method is based on the Piecewise Aggregate Approximation (PAA) representation [12, 13]. Many studies have proposed improvements to SAX. These studies [9, 14-18] have shown that the SAX method is still an open field of research and that new developments are required to bring new ideas to improve the performance of SAX.

The latest research aimed at improving the SAX algorithm is by [19] and focuses on indexing and mining time series data. However, this research has not led to algorithms that can be scaled to the increasingly massive datasets that are encountered in science, engineering, and business. The iSAX shows how a novel multi-resolution symbolic representation can be used to index datasets that are several orders of magnitude larger than any datasets that have been considered in the literature. This approach allows for both fast exact searches and ultra-fast approximate searches. This scenario shows how both types of searches can be combined and exploited; running both searches as sub-routines in data mining algorithms allows for exact mining of truly massive real-world datasets, which contain millions of time series. Because this approach is used to enhance the SAX representation in terms of indexing and does not account for both the alphabet and word sizes, we decided not to compare our method with this approach. Otherwise, the comparison would be unfair.

Another algorithm for time series representation is proposed by [20]. The algorithm is using Bag of Patterns (BOP). It works as follows. For each time series, it uses a sliding window and extracts every possible subsequence of length n (a user-defined parameter). Each subsequence is normalized to have a mean of zero and standard deviation of one before it is converted to a SAX string. As a result, it obtains a set of strings, each of which corresponds to a subsequence in the time series. As noted in SAX, given a subsequence Si, it is likely to be very similar to its neighboring sub sequences, Si-1 and Si+1 (i.e. those that start one point to the left, and one point to the right of Si), especially if Si is in the smooth region of the time series. These sub sequences are called trivial matches of Si. To avoid over-counting these trivial matches as true patterns, it needs to perform numerosity reduction. Since SAX preserves the general shape of the sequence, in some cases it might see that multiple

consecutive sub sequences are mapped to the same string. In that case, the algorithm records only the first occurrence of the string, and ignores the rest until it encounters a string that is different. In other words, for each group of consecutive identical strings, it records only the first occurrence and count this group of occurrences only once. The proposed algorithm shows outperforms the existing methods in clustering, classification, and anomaly detection on several real datasets.

The majority of the methods used in data mining in time series assume that the time series are discrete. Nevertheless, most applications generate and use floating-point data. Therefore, there are numerous approaches to the discretisation of time series using floating-point values. Different approaches and different measures of function have been proposed, such as in [21]; the purpose of this method is to detect the persistent states of the time series, and the method does not require parameter specification. The applicability of this method is restricted to the existence of persistent states in the time series; this is uncommon in most of real-world applications. This method processes a single time series at a time, so the discretisation criterion is not generalized to the complete dataset. In [16], time series values are represented as a multi-connected graph. Using this representation, similar time series are grouped in a graphical model. However, this method is limited in that it only works with one series at a time. Generating alphabet and word sizes becomes a vital challenge in discretising and reducing time series in the data representation phase. Hence, much of the research involved in using time series data sets focuses on solving these problems.

Acosta et al. [22] introduced the entropy based linear approximation (EBLA2) algorithm; this algorithm was proposed as a method to automatically generate the alphabet and word sizes to maximize classification accuracy. Similar to a greedy search, a heuristic approach is used to lead to a specific solution that is obtained from the best result of iteration. EBLA2 requires no parameters for the search because the outcome is deterministic.

An enhanced version of the algorithm EBLA2 was proposed in [23] and is called EBLA3. This enhanced algorithm performs a broader search than does EBLA2 while using simulated annealing as the discretisation scheme for temporal datasets. This algorithm automatically generates parameters with which it is possible to identify better discretisation schemes. However, the results obtained by this approach may be improved by enabling entire populations to generate discretisation schemes; this strategy may quickly reach good solutions.

Daniel et al. [6] proposed a new algorithm for time series discretisation using an approach that applies a genetic algorithm called GENEBLA. This algorithm generates random alphabet and word sizes for given time series. The GENEBLA has proven to be an efficient search algorithm for the state space or an approximate solution optimization. It also performs efficiently when the user does not have precise domain expertise because genetic searches possess the ability to explore and learn from their domain. Several other GA-based algorithms used for discretisation are discussed in [24].

Bakar et al. [18] proposed an improvement to the time series data discretisation approach by using the relative frequency and K-nearest neighbor functions, which is called the RF method. The main idea of the RF method is to improve the process of determining a sufficient alphabet size while discretising the time series data. The proposed approach improves the time series data representation in the SAX representation. Alphabets are represented as symbols and can ensure an efficient mining process, whereas better knowledge in the model may be obtained without major knowledge loss. The basic idea is not to minimize or maximize the size of the alphabet for temporal patterns over their class labels. Thus, RF can improve the representation precision without losing the symbolic nature of the original SAX representation.

Another proposed algorithm [17] for the SAX, seeks to efficiently recognize and accurately discover important patterns that are essential for time series data. The proposed improved SAX, called iSAX, includes the relative frequency and K-nearest neighbor (RFknn) algorithm. The main task of the iSAX algorithm is to determine the number of intervals represented in symbols (alphabet size) that is sufficient to ensure efficient mining and a good knowledge model without major loss of knowledge. Thus, iSAX can improve representation precision without losing the symbolic nature of the original SAX representation. The iSAX algorithm is compared with the original SAX and PAA representations and demonstrates its improvement in quality of the model. Ten time series rainfall data sets were used. The experimental results showed that iSAX yields better representation terms and minimal Euclidean distance.

Ahmed et al. [9] proposed an improved SAX representation, known as HSAX, which uses the Harmony Search algorithm (HS) to explore the optimal alphabet (a) and word size (w) for a SAX representation. The HS algorithm was developed to maximize information rather than to improve the error rate. The HSAX algorithm was applied to standard time series datasets. The experimental results were compared with those of other meta-heuristic methods such as GENEBLA [6] and the original SAX algorithm. These results show that HSAX generates larger word and alphabet sizes than GENEBLA and SAX. HSAX also achieves an error rate that is lower than that of SAX and comparable to that of GENEBLA.

Another algorithm for SAX representation was introduced by Rechy-Ram [8]. The main idea in this work is that the algorithm includes optimization of alphabet size and word size. This algorithm uses evolutionary programming (EP) and continues the search for an efficient discretization scheme by using a fitness function which considers three criteria: the entropy with respect to the classification, the complexity measured as the number

of different strings needed to represent the complete data set, and the compression rate, which is assessed as the length of the discrete representation.

Many studies have proposed improvements to SAX. These studies [9, 14-18] have shown that the SAX method is still an open field of research and those new developments and ideas are needed to further improve the performance of SAX. Table 1 shows the methods that are proposed to improve SAX in terms of alphabet and word sizes.

Table 1: The main focus of SAX algorithm variants

Algorithm	Author	Using	Data	Main focus
ELBA2	Acosta (2005)	Grady search	UCR(20)	Minimisation
ELBA3	Alejandro et al. (2007)	SA	UCR(22)	Minimisation
GA	Acosta and Lopez (2009)	GA	UCR(22)	Minimisation
RF	Bakar et al. (2010)	Relative Frequency	UCR(22)	Maximisation
HSAX	Ahmed et al. (2011)	Harmony search	UCR(22)	Maximisation
EP	Rechy-Ram (2011)	Evolutionary Programming	UCR(20)	Minimisation

The proposed algorithms for improving the SAX time series representations are required to minimize the complete original set of time series into a smaller set of time series. For example, the size of the word is found by separating intervals (of given lengths) of the time series; the number for the alphabet size reduces the number of distinct values in the series. Table 1 shows that all of the methods that have attempted to improve the SAX method focus only on minimizing the time series rather than on maximizing it. Thus, these methods may lead to large reductions of the time series, which, in turn, results in the loss of important information.

Hence, we must analyze temporal data to obtain an efficient discrete representation of the given datasets. We must take the problems described above into account as one part of the search; however, our focus is mainly to design an algorithm that maximizes both the word size and the alphabet size for given datasets. These considerations have yet to be discussed in the proofs of proposed algorithms; often, researchers attempt only to compress datasets and improve their classification accuracy. This may lead to a loss of some information that is both important and sensitive; for example, applying these algorithms in weather and financial applications, in which the details of the information in the data should not be removed, may lead to knowledge loss. Removing these details may result in the destruction of essential information and hidden patterns.

III. Proposed Methods

Time series data are produced, continued, and processed within a wide range of application domains in various fields such as economics, engineering, science, medicine and sociology. In this section, we present an improved SAX algorithm to represent time series data called SAX^{++}. The SAX algorithm works with two parameters: alphabet size and word size. Specifically, we propose an integration of the two algorithms RF and HS^{MPAR} to improve SAX. Using these algorithms allows for the integration of a large amount of information through maximization of the alphabet size and word size while maintaining a low error rate.

3.1 Proposed RF method for alphabet size

In this section, we review the RF binning approach that is proposed in [18]. The main goal of RF is to optimize the alphabet size and minimize changes and losses in knowledge from the original information. It is important to achieve these goals, especially in a dataset that requires the means and the integrity of the data to be retained. The principle of RF and the distance measure are used to determine a sufficient number of intervals (alphabet size) that can ensure an acceptable to excellent knowledge model, which will be obtained without a major loss of knowledge. The explanation of RF is presented in the following:

Definition 1: Let $C=c_1,c_2,c_3,..,c_n$, where C is a time series and n denotes the time series length.

Definition 2: Let $a=a_1,a_2,a_3,\ldots,a_m$, where a denotes alphabet size and m denotes the number of observations in C.

$$a_{rf} = \sum_{j=1}^{m} a_j \frac{1}{\sum_{j=1}^{m} a}, where, j = 1..m. \quad (1)$$

Where $j=1,2,\ldots,m$, and a_{rf} is the relative frequency for each alphabet (a) in a time series C and a_j is the alphabet in C.

$$T = \sum_{j=1}^{m} \frac{a_{rfj}}{\sum_{j=1}^{m} a_{rfj}}, where, j = 1..m. \quad (2)$$

Where T, is the cumulative relative frequency of alphabets in C, which is known as the threshold value.

$$x_{class}(a_j) = \begin{Bmatrix} c, a_{rfj} \geq T \\ \dot{c}, a_{rfj} \leq T \end{Bmatrix}, where, j = 1..m. \quad (3)$$

Definition 3: Let $X = \{C, \dot{C}\}$ where C denotes the class high confidence alphabet and $\dot{C}$ denotes the class low confidence alphabet.

Definition 4: Let $a=\{a_{1class}, a_{2class}, a_{3class}, .., a_{wclass}\}$ where a denotes the alphabets with the class label equal to the high confidence alphabet and w denotes the number of observations.

Definition 5: Let $a' = \{a'_{1class}, a'_{2class}, a'_{3class}, ..., a'_{kclass}\}$ where a' denotes the alphabets with the class label equal to the low confidence alphabet and k denotes the number of observations.

$$Dist(a_i, a'_j) = \min im \sum_{j=1}^{k} \sum_{i=1}^{w} (a_i - a'_j)^2 . \quad (4)$$

Where the *Dist* is *a* distance function, which computes the difference between the two alphabets *a* and returns the minimum values for the distance.

$$\hat{a} = \{a, Dist(a_i, a'_j) \equiv \min im. \quad (5)$$

Where $\hat{a}$ is denotes the new alphabets and *w* is a new number for the generated alphabet size.

$$\hat{a} = (\hat{a}_1, \hat{a}_2, \hat{a}_3, ... \hat{a}_w, \}. \quad (6)$$

In Eq. (1), the RF algorithm starts to compute the relative frequency for each observed alphabet in *C*. The RF means that each alphabet's frequency value is divided by the number of observation alphabets. Then, Eq. (2) shows how the algorithm computes the threshold values (*T*), which relies on the summation of *rf* for each alphabet and is divided by the number of observations. In Eq. (3), RF shows how the class is assigned to each alphabet: either C (high confidence) or $\dot{C}$ (low confidence). Moreover, RF continues to find the distance between the two alphabets as shown in Eq. (4). Lastly, RF merges the two alphabets based on the minimum values that are yielded by Eq. (5), and lastly, Eq. (6) demonstrates the new alphabets of the new alphabet size.

Previous discussions have indicated that to eliminate a redundant alphabet size (intervals), state-of-the-art interval selection methods must rely on using subset evaluation, which substantially reduces the number of intervals and leads to greater information loss. Hence, this method can produce better results than can methods that do not address information loss. Our goal is to efficiently find the optimal set of alphabet sizes using the RF algorithm and to integrate it with the HS^{MPAR} algorithm to find word size. In the next section, we will discuss the HS^{MPAR} algorithm.

3.2 Proposed Harmony Search with Multi-pitch Adjusting Rate (HSMPAR) for word size.

In this section, we explain our proposed HS^{MPAR} algorithm. It is based on the basic HS algorithm with Multi-pitch Adjusting Rate as well as five important steps outlined below. HS^{MPAR} algorithm is integrated with RF algorithm to calculate the optimum alphabet and word size .Our technique takes the order of the information into account without changing the original information. The basic HS^{MPAR} algorithm consists of the following five steps: i) parameter initialisation; ii) harmony memory initialisation; iii) new harmony improvisation; iv) harmony memory update; and v) termination criterion check.

Step 1. Parameter Initialisation:

Parameter initialisation: In the first step, the optimisation problem is specified using the following decision variables and other variables. $F(x)$ is the fitness function, x_i is the decision variable representing two variables which are alphabet size (a_i) and word size (w_i), and *HMS* represents the candidate solutions for each variable (x) as a variable value word size (w).

In this study, there are 50 randomly selected candidate *HMS* values which are obtained from *a* and *w* for each variable, where *N* is the number of decision variables, *k* is *HMS*, and x_i is the random solution. *HMCR*=0.2 and *PAR*=0.9. *HM* is shown in Eq. (7):

$$HM = \{x_{ij}(1), x_{ij}(2), x_{ij}(3), ..., x_{ij}(HMS). \quad (7)$$

Here, *HMS* is the number of candidate values for the discrete decision variables, and x_{ij} is the decision variables as in Eq. (7).

$$F(X) = \{f(x^1), f(x^2), f(x^3), ..., f(x^{HMS}). \quad (8)$$

Here, $f(x^1), f(x^2), \ldots, f(x^{HMS})$ shows each solution vector for design variables and the corresponding fitness function value.

The HS algorithm parameters are also specified in this step: *HMS* (harmony memory size = the number of simultaneous solution vectors in harmony memory), *HMCR* (harmony memory considering rate), *PAR* (pitch adjusting rate), and the number of improvisations (number of fitness function evaluations). These algorithm parameters are presented in Table 2.

Table 2. Parameters description

Parameter	Description
F(x)	fitness function
Xi	decision variable (*W*, *a*)
N	number of decision variables
HM	harmony memory
HMS	number of solutions in HM
PAR	patch adjustment
HMCR	harmony memory consideration
P	probability
W	word size
A	alphabet size
Ni	number of iterations
N	time series length

Step 2.Harmony Memory initialisation

In Step 2, the Harmony Memory (*HM*) is crammed with as many randomly generated solution vectors as the size of the *HM* will allow. The harmony memory initialisation consists of the following steps: Every harmony *i* randomly generates the solution vector x_1= $\{w_1,w_2,...,w_{HMS-1}, w_{HMS}\}$ x_2= $\{a_1, a_2, ..., a_{HMS1}, a_{HMS}\}$ where $\{2\leq w<n\}$, and *HMS* is the number of solutions in *HM*. The random values generated range between [0,1], where the random values represent the real values of the time series sequences. The information gain is then calculated based on entropy for $F(x)$.

Step 3: New Harmony Improvisation

Basically, a new harmony vector, $x' = (x'_1, x'_2,..., x'_{HMS})$ is improvised by following three rules: (i) random selection, (ii) *HMCR* consideration, and (iii) multi-pitch adjustment.

(1) Random selection

When HS^{MPAR} determines the value $_{il}$ for the new harmony $x' = (x'_1, x'_2,..., x'_{HMS})$, it randomly picks any two values from total value range $\{x_i(1),\ldots,x_i(HMS)\}$ with a probability of (1-*HMCR*).

(2) HMCR consideration

When *HS* determines the value $_{i,}$ it randomly picks any two values from $HM= \{x_i (1),\ldots,x_i (HSM)\}$ with the probability *HMCR*. The probability of *HMCR* can be calculated using the uniform distribution $U(0, 1)$:

$$P_{HMCR} = \text{int}(U(0,1), HMS) + 1. \quad (9)$$

As the musician plays any pitch out of the preferred pitches in his/her memory (for example, *k* candidates is the number of random solutions (*a*,*w*) for each variable $HM=\{k_1, k_2, k_3, \ldots, k_{HMS-1},k_{HMS}\}$, the value of the decision variable x'_i is chosen from any number of pitches stored in HM with the probability P_{HMCR} while it is randomly chosen as shown in Eq (9).

(3) Multi-pitch adjustment

After the value of $_i$ has been randomly picked from the *HM* in the above memory consideration process, it can be further adjusted into neighbouring values by adding a certain amount to the value, with the probability *PAR* as shown in Eq. (10). For example, the value of k_1 can be adjusted to $(k\pm m)$ with respect to $(1<k<n)$, with a range of time series original length $(U(2, n-1))$, while the original pitch obtained in the *HM* consideration is maintained at the probability PAR.

$$P_{PAR} = \text{int}(U(0,1), HMS) + 1. \quad (10)$$

In this study, a new harmony vector represented by word size and alphabet size, x_1= $\{w_1,w_2,...,w_{HMS-1},w_{HMS}\}$ and $x_2=\{a_1, a_2, ..., a_{HMS1}, a_{HMS}\}$ is improved by the three rules with the following steps.

1. Select two random solutions $x([w_{rndm1},a_{rndm1}]$ and $[w_{rndm2},a_{rndm2}])$.
2. The harmony memory consideration rate is the intersection of the two solutions of the probability *HMCR*. For example, (w_{rndm1},a_{rndm1}) is swapped with (w_{rndm2},a_{rndm2}) and $P_{HMCR}>0.2$, then the *HMCR* will be (w_{rndm1},a_{rndm2}) and (w_{rndm2},a_{rndm1}).
3. The Harmony Pitch Adjustment *PAR* is converted more than once (multi-pitch) using the *PAR* probability. If the $P_{PAR}<0.9$, then the random value is generated in the range of the original time series $(U(2,n-1))$. For example when $k=i$, then *k* is converted to $i\pm (U(2,n-1))$.

Step 4: Harmony Memory Update

If the new harmony $F(x_{randm1})$ and $F(x_{randm2})$ are better than both the selected old harmony $F(x_1)$ and $F(x_2)$ in the *HM*, as judged by the fitness function value, the new harmony values are included in the *HM* and the existing worst old harmony values are excluded from the *HM*.

1. For each new harmony, x_1 and x_2 *are* adjusted by *PAR* ($solution_1$, $solution_2$).
2. Calculate the fitness function for each *x*, *F(x)*.
3. Repeat until the new solutions (x_i) get better than the randomly selected solution in *HM*.
4. Include the new solutions in *HM* and exclude the old solutions from *HM*.
5. End

Step 5: Termination Criteria Check

Termination criterion check: If the termination criterion (the number of improvisations) is reached, the computation is stopped. Otherwise, *Steps 3* and *4* are repeated.

Fitness Function

Any optimisation algorithm requires a fitness function to measure the fitness of the set of solutions that are created. Most discretisation techniques require a heuristics approach to avoid the a priori definition of alphabet size and word size[15, 25]. Multi-Interval Discretisation (Ent-MDLP) is a discretisation technique that uses the entropy measure proposed by Fayyad and Irani. Ent-MDLP uses the entropy minimisation heuristic (EMH) to divide continuous values and the minimum description length criteria to control the number of intervals produced in continuous space. The stopping criterion for this technique is the minimum description length principle (MDLP). Later, the MDLP is applied in many other techniques and is known as one of the best discretisation techniques that exist.

A new quality score for meaningful, unsupervised discretisation of the time series [21] is proposed by accounting for temporal information while searching for persistence. Recently [22], a fitness function to select the optimal cut points that are based on the information gain measure was proposed. Fitness functions should reflect how well a set is represented by a set of features. Hence, we make use of the information gain measure as a fitness function for a given solution or individual via the harmony algorithm. Formally, the maximum value of entropy is based on information gain and can be stated as Eq. (11) where *Sclass* and *Svalue* are the time series class and value, respectively. [6] The variable #*Svalue* is the number of time series with a value in *S,* and #*Sclass* is the number of time series in class *S*. The entropy of *Sclass*, Ent(*Sclass*) is given by Eq.(12). The term *c* is the number

of classes and p_i is the probability of class i in *Sclass*. It is important to note that the entry time series is considered to be one attribute and that the discretisation scheme is considered for the entire dataset. This method allows the algorithm to find a good global solution that maximises the entropy of the data.

$$Gain(Sclass, Svalue) = Ent(Sclass) - Ent(Svalue). \quad (11)$$

$$Ent(Sclass) = -\sum_{i=1}^{c} p_i \log_2 p_i. \quad (12)$$

The entropy of S when it takes on the time series value *Svalue* is given by Eq. (13), and the fitness function is given by Eq.(14) where n is the time series length and $gain_i$ is the gain between time series values and its class, $i=1...n$.

$$Ent(Svalue) = p(S \mid v)_i \log_2 p(S \mid v)_i \quad (13)$$

$$F(x) = \frac{\sum_{i=1}^{n} Gain_i}{n}. \quad (14)$$

3.3 Proposed SAX++ method

There are three critical factors that demonstrate the major difference between the HSAX that was reported previously [9] with our proposed SAX^{++} algorithm. First, in [9], HSAX is implemented only to generate the word size based on one pitch adjustment at each iteration; SAX is then run using a predefined alphabet size. In contrast, our proposed HS^{MPAR} generates both alphabet size and word size and it works with multi-pitch adjustment (four pitches at one iteration). Secondly, the previous HS runs with 10 populations and 50 iterations, whereas in this study, the HS^{MPAR} extends the number of individuals to 50 populations, and the number of improvements is increased to 100 iterations. Thirdly, in this study, whenever the HS^{MPAR} returns the optimal alphabet size, it will pass over to the RF process to maintain it. It then returns the most frequent alphabets to the HS^{MPAR} process. Our algorithm aims to find solutions for given time series, which is the problem described in the following definition:

Definition 6: Let $D=\{C_1,C_2,C_3,...,C_N\}$, where D denotes the time series dataset, $C=\{c_1,c_2,c_3,..,c_n\}$,where C denotes the time series, n denotes the length of the time series, $i=1..n$.

Input: $C=c_1,c_2,c_3,...,c_n$, *a time series*
HMCR=0.2
PAR=0.
Output: Alphabet size and Word size

1:*Begin*
2:*Initialise the Harmony Memory (HM),*$HM=([w_1,a_1],[w_2,a_2],[w_3,a_3],...,[w_{HMS-1},a_{HMSI}],[w_{HMS},a_{HMS}])$,*: Calculate fitness for each solution vector generated in HM.* $F(X)=(x_1,x_2,x_3,..,x_{HMS-1},x_{HMS})$,
3:*WhileiGeneration<nGeneration*
4: *For i=1 to number of decision variables N do where N=1..2*
5: $P_{HMCR}=Rand(0,1)$, $P_{PAR}=Rand(0,1)$.
6: *If* (P_{HMCR}<HMCR) */*(memory consideration)*/*
7:*x[i]=Fselection (HM) /*two solutions will be randomly chosen from harmony memory*/*
8: $\grave{x}[i] =$ *FHMCR(x[i]) /* harmony memory consideration rate*
9: *If* (P_{PAR}< *PAR) /* pitch adjustment)*/*
10: $\grave{x}[i] = \grave{x}[i] \pm U(2,n-1)$*/* pitch adjustment rate * four */*
11: *End if*
12: *Else*
13: $PAR=P_{PAR} \pm rand(0,1)$; $HMCR=P_{HMCR} \pm rand(0,1)$
14: *End if*
15: *End for*
16: $fitness\_x = Ffitness(\grave{x})$.
17: *For each* HM_j *in HM do, where j=1..HMS*
18: *If the new solution is better than the random selected solution in the harmony memory*
19: *then Replace the worst solution by the new one*
20: *End if*
21: *End for*
22: *Increase(iGeneration)*
23: *End while*
24: *Return best individual (population).*

Algorithm 1: Phase 1- HS^{MPAR} Algorithm for optimal alphabet and word size

The HS^{MPAR} is designed to solve the problem of discretisation. As discussed earlier, the problem of discretisation of a time series relates to finding the number and range of intervals of the time series that maintain relevant information (alphabet size). It also relates to the discretisation of data on each interval of time (word size). In this method, $C=\{c_1,c_2,c_3,...,c_n\}$ is a time series and n is the length of the time series. The algorithm starts with initialising the Harmony Memory (*HM*), where *HM* includes random solutions, and each harmony contains two decision variables, x_1 *and* x_2, which denote the alphabet and word size, respectively. These solutions are randomly generated and follow the hard rule that states that ($2 \geq w < n$) and ($2 \geq a < w$). The P_{HMCR} contains the probability values between [0, 1] and the P_{PAR} contains the probability values between [0, 1]. *nGeneration* is the number of iterations in the algorithm. The efficiency of the algorithm is the amount of improvement. It is based on the harmony memory consideration and random selection, where two random solutions are picked from *HM*. For instance $HM_1(w_1, a_1)$and $HM_2(w_2, a_2)$ may be used. The HS^{MPAR} algorithm then tries to improve both solutions at the same time, first if the P_{HMCR} is less than *HMCR,* then the harmony memory-considering rate function (FHMCR()) is called to improve the selected solutions. For instance, $HM_1(w_1, a_1)$ and $HM_2(w_2, a_2)$ may be swapped similarly to cross over in GA, where $HM1(a_1)$ is swapped with $HM2(a_2)$ to result in $HM_1(w_1,a_2)$ and $HM_2(w_2, a_1)$. In the last stage of the improvement, if the P_{PAR} satisfies the *PAR*, then the pitch adjustment rate is performed four times (multi-pitch adjustment) to improve the selected solutions, where *PAR* is adjusted by adding or minimising a random value in the range of $U(2,n\text{-}1)$ and $U(2,w)$. Finally, the improvement of solutions in *HM* is measured using the fitness function, $F(x)$. If the new solutions are better than

the selected old solutions in *HM*, then the old solutions from *HM* are excluded and the new solutions in *HM* are included to return the best individual.

Example

A given time series of length n=90 and a=13 are passed through the HS^{MPAR} algorithm. As explained earlier, each time series has the chance to be run 10 times with *nGeneration*=100 iterations for each time the series run. The number of populations is *HMS*=50, the P_{HMCR}=0.2 and the P_{PAR}=0.9. First, the original time series is tested with $F(x)$ and the output is $F(x)$ =0.63.

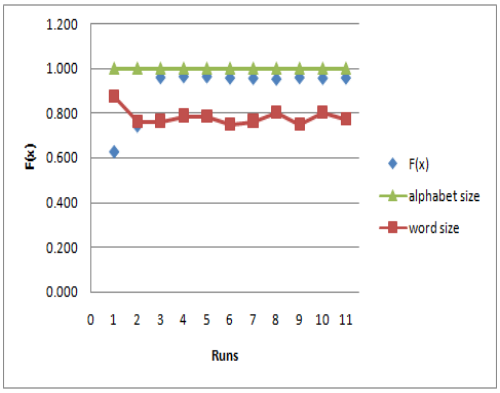

Fig. 1a.

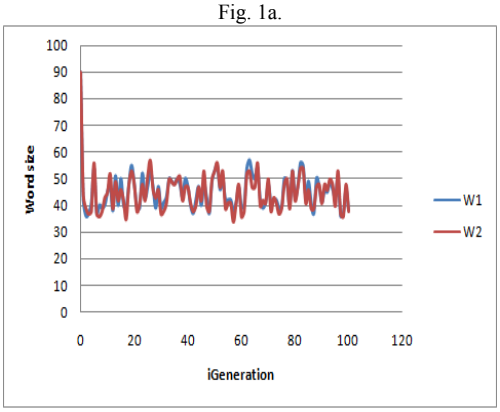

Fig. 1b.

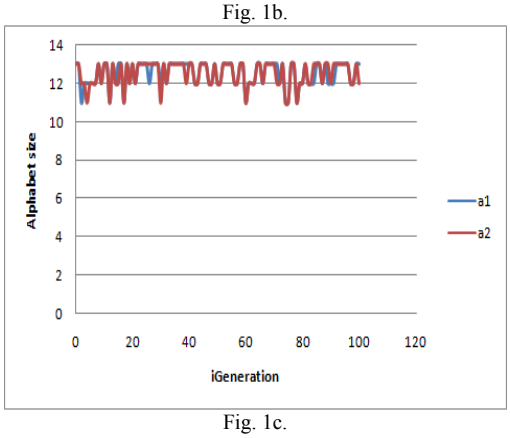

Fig. 1c.

Fig 1: Two random optimal solutions that were generated after 100 iterations.

Figure 1a shows the improvement of the HS^{MPAR} solutions, including the solutions for (F (x)) that are returned after 100 iterations the solutions after 10 applications of HS^{MPAR}. *Solution1* reaches the best gain at 0.957, and *solution2* reaches the best gain at 0.955. It should be noted that *solution1* starts at an initial accuracy $F(x)$ =0.62 and is improved with 100 iterations and ten runs. Through this improvement, *PAR* is adjusted four times at each *iGeneration*. Figure 1b represents the number of word sizes that was generated at each *iGeneration*, associated with the best $F(x)$ at application number 6. As shown in figure 1c, the best word size (47) is reached at $F(x)$=0.957, and the best alphabet size (12) is achieved at the same $F(x)$. Table 3 shows an example of the first 10 populations in *HM* that are used in this example.

Table 3. Example of HM

Populations	Word size	Alphabet size	F(x)
1	42	12	0.60
2	51	12	0.69
3	46	12	0.63
4	38	12	0.73
5	48	11	0.63
6	50	13	0.67
7	45	13	0.62
8	46	13	0.68
9	40	11	0.64
10	45	11	0.67

Input: $C=c_1,c_2,c_3,...,c_n$, ***a time series Alphabet size***
Output:Alphabet size

1:Begin
2: For each c_i in C do, i=1..n
3: $\dot{C}$ = FSort(C),sorting the values of C
4: $\dot{C}_{observed}$ = FGenerateAlphabet($\dot{C}$,a), a=1..m.
5: End For
6: For each $\dot{C}_{observedi}$ in $\dot{C}_{observed}$ do, i=1...m.
7: For each C_j in C do, j=1..n.
8: $Rf_{observedi}$=FRelativefrequency($\dot{C}_{observed}$,C).
9: $Rf\dot{C}_{observedi}$ =FCumulativeRF=($Rf_{observedi}$).
10: $\bar{a}_{Rfi}$ = FCumulativeRF=($Rf_{observedi}$).
11: Threshold=Fmean($\bar{a}_{Rfi}$,m)
10: End for
11: End for
12: For each $Rf_{observedi}$ in $Rf_{observed}$ do, i=1...m .
13: For each $\bar{a}_{Rfi}$ in $\bar{a}_{Rf}$ do, j=1...m
14: If $\bar{a}_{Rfi} \geq$ Threshold then
15: $Rf_{ClassHighi}$= $\dot{C}C$,High_confidenceinterval class
16: Else
17: $Rf_{ClassLowi}$=CC,Low_confidenceinterval class
18: End if
19: End for
20 End for
21: for each $Rf_{ClassLowi}$ in $Rf_{ClassLow}$ do, i=1..m
22 for each $Rf_{ClassHighj}$ in $Rf_{ClassHigh}$ do, j=1..a
23: D(minim)= Fdistance($Rf_{ClassLowi}$,$Rf_{ClassHighj}$).
24: MergeAlphabet($D_{,observedJ}$,$Rf_{ClassLowi}$,$Rf_{ClassHighJ}$).
25: End for
26: End for
27: End

Algorithm 2: Phase 2- RF Algorithm

In this phase, RF is used to maintain the alphabet size of the given time series, and the time series C is restructured based on the Frequency method. The algorithm starts by sorting time series C, via the Fsort function with the limit that the function maintains the order of original time series. Then, the FGenerateAlphabet function is called to generate the number of alphabets based on (a) that are derived from the HS^{MPAR} algorithm. Next, the FRelativefrequency function computes the largest number of frequency alphabets (intervals) and returns each alphabet with its corresponding frequency value. Then, these alphabets are passed through the FCumulativeRF process to compute the relative value for each alphabet as the number of observations divided by the number of elements in $C(n)$. The Fmean function returns the mean of these relative values, and the mean is considered to be the Threshold value. If the relative value of the alphabet is higher or equal to the Threshold value, the alphabet class is assigned to the *High_confidence* class ($\dot{C}C$). If the relative value of the alphabet is not greater than or equal to the Threshold value, it is assigned to the low confidence class (CC). Finally, the distance function is used to measure the distance between two objectives (alphabets), i.e., alphabets assigned to the high confidence class and alphabets assigned to the low confidence class. The Fmerge function is then applied to merge two alphabets that are close in distance, which is based on the lowest similar value of the distance function. *RF returns* the new number of alphabets.

Input: $C=c_1,c_2,c_3,...,c_n$, ***a time series***
Output: $\hat{c}=\hat{c}_1,\ldots,\hat{c}_W$***=: A symbolic representation of a time series***

1: *Begin*
2: *For each* C_K *in D do, where k=1..N*
3: *X(w,a)=FHS(*C_k*);*/*Call* HS^{MPAR} algorithm function to compute the alphabet and word size*/)
4: $\overline{a} = FRF(C_k, a)$;/*Call RF algorithm function to compute the alphabet size again*/
5: *For each* c_i *in* C_K *do, where i=1..n,k=1..N*
6: $\bar{C}_{PAA} = \frac{w}{n} \sum_{i=\frac{n}{w}(1-i)+1}^{\frac{n}{w}i} c_i$;/*Call PAA algorithm to reduce time series length */
7:*End For*
8: $\bar{C}_{normlz} = FNormlaz(\bar{C}_{PAA})$
9: *For each* C_i ***in*** C_k *do, where i=1..w,*
10: *For each* β_j ***in*** β *do, where j=1..L,where L=1..50;*
11: $\hat{C}_i = alpha_j$ ***if*** $\beta_{j-1} \leq \hat{C}_i < \beta_j$
12: *End if*
13:*End for*
14: *End for*
15: *End*

Algorithm3: Phase 3-SAX^{++} algorithm

The main contribution of the work is stated in Phase 3 of the Algorithm3, which tries to settle the problem of discretisation in the SAX^{++} procedure via finding the optimal alphabet size (a) and word size (w) for a given time series. *FHS* is a function that receives a time series of length n. Through this function, the HS^{MPAR} algorithm searches the problem space ($2,n$) and returns the best individual population (w, a).The steps are explained in Phase 1 of Algorithm1. The *FRF* function is then applied; the process of this function is based on the RF algorithm. It aims to maintain the output of the HS^{MPAR}. RF receives a time series length of n with the optimal alphabet size defined by the HS^{MPAR} algorithm. The RF function then determines the optimal alphabet size by testing the frequency of the HS^{MPAR} alphabets. The output of RF either confirms or improves the alphabet size of the HS^{MPAR}. The RF process is explained in Phase 2 of Algorithm2. The PAA algorithm is employed to reduce the time series length from *length n to length w* where firstly, PAA receives word size (w) via the HS^{MPAR} algorithm and then aggregates the time series based on step 4 of Algorithm3. The output of PAA is $PAA = \check{C}_1,\ldots,\check{C}_W$. *PAA* is explained in section 2. Normalisation of the time series then takes place as denoted in Step 6 of Algorithm 3.

The FNormlaz function receives aggregated time series via PAA and returns the normalised time series that have high Gaussian distributions and can simply determine the "breakpoints" that will produce a equal-sized areas under the Gaussian curve[26]. The output of *FNormlaz* is $\bar{C}_{normlz}$. The discretisation and transformation processes start using this output. The discretisation process is performed based on the lookup table; it is explained in [4], for a given time series with an alphabet. The transformation of the time series to the symbolic representation is then accomplished depending on the number defining the alphabet size. The output of SAX^{++} is produced with the symbolic representation of a new time series, $\hat{c}= \hat{c}_1,\ldots, \hat{c}_W$.

IV. Experiments

Experiments were conducted in which RF and HS^{MPAR} were integrated into the SAX time series data representation to generate sufficient alphabet and word sizes. The performance of SAX^{++} was evaluated by the alphabet and word sizes and the error rates on twenty benchmark time series datasets that were obtained from time series datasets website. The SAX^{++} algorithm was compared to the original SAX and GENEBLA algorithms.

4.1 Experimental design

The performance of the SAX^{++} method was tested using 20 datasets that are available on the time series classification/clustering web page[27]. These datasets were previously used by [5] to test the SAX approach. The SAX and PAA were used in the experiment. The RF and HS^{MPAR} algorithms were applied to determine an efficient number of intervals for the alphabet size (a) and the word size (w). Then, SAX was run with the same generated parameters (generated by the *RF* and HS^{MPAR} algorithms) of the alphabet size (a) and word size (w), which is called SAX^{++}. SAX^{++} was compared to the original Eumonn's 1-NN (1-NN EU) and 1-NN SAX [5] and GENEBLA [15]. The alphabet size produced by RF and HS^{MPAR}, denoted as *a-*SAX^{++}, was compared to the

fixed alphabet size *a*, which was given by the original SAX and is denoted as *a-SAX* [5] and compared to the alphabet size, which is denoted as *a-G*.

Word size was produced by HS^{MPAR} and is denoted as *w*-SAX^{++}. It was compared to the fixed word size given by Keogh's experiment, which is denoted as *w-SAX*. We then compared it to the word size that was generated by GENEBLA, which is denoted as *w-G*.

We used 1-NN to evaluate the SAX^{++} method, and we also used Absolute Distance for the classification performance. The classification performance was evaluated using the raw data (the continuous time series). It is important to note that SAX uses its own similarity measure[5]. When using algorithms and parameters for classification as described above, a lower error rate indicates better classification performance. This experiment was conducted to determine whether SAX^{++} can maintain more alphabet and word sizes compared to the previous approaches to the time series problem.

4.2 Results

The experimental results showed that the proposed SAX^{++} algorithm performs better in terms of error rates returned in several datasets when compared to the 1-NN EU technique. SAX^{++}yielded exceedingly low error rates when compared to the original SAX algorithm in most datasets that have larger alphabet and word sizes, as shown in Figure 2 and Figure 3. SAX^{++} performed more efficiently on more than 70% of the datasets with respect to the returned error rates, as shown in Table 4.

Table 4 shows that when compared to GENEBLA, SAX^{++} approached similar error rates using different parameters (*the alphabet size* and *word size*), with the advantage that SAX^{++} did not require these parameters a priori. A priori parameters were not required because they are automatically calculated in SAX^{++}, whereas in SAX, a high error rate was returned with fixed alphabet and word sizes. In some datasets, SAX^{++} provides better performance compared to the original SAX. It should be noted that SAX^{++} yielded lower error rates on 9 datasets, meaning that it is more effect than SAX and GENEBLA. SAX^{++} was similar to the other two algorithms in four data sets; in addition, SAX^{++} performed more poorly compared to SAX and GENEBLA in seven datasets.

Table 4. Error rates obtained by SAX^{++}, GENEBLA and SAX

No	Data sets	SAX^{++}	GA	SAX	p-value SAX^{++}
1	coffee	0.398	0.420	0.464	0.001
2	Adiac	0.502	0.490	0.890	0.004
3	Gun Point	0.290	0.200	0.180	0.001
4	ECG200	0.320	0.200	0.120	0.005
5	Beef	0.400	0.500	0.567	0.001
6	Olive Oil	0.533	0.380	0.833	0.00
7	Fish	0.200	0.320	0.474	0.001
8	Trace	0.450	0.500	0.460	0.005
9	FaceFour	0.185	0.210	0.170	0.001
10	50word	0.300	0.44	0.341	0.005
11	faceAll	0.299	0.330	0.330	0.001
12	Swedish Leaf	0.390	0.400	0.483	0.004
13	Two pattern	0.033	0.280	0.081	0.001
14	Wafer	0.038	0.010	0.020	0.005
15	Yoga	0.195	0.200	0.195	0.001
16	Light7	0.397	0.490	0.397	0.00
17	Light2	0.203	0.200	0.213	0.001
18	Osu Leaf	0.399	0.500	0.467	0.005
19	Control chart	0.100	0.310	0.467	0.000
20	CBF	0.102	0.100	0.104	0.000

Table 4 indicates that the proposed method, SAX^{++}, achieved high significance (p-value) in all data sets. SAX^{++} was compared to the GA and SAX methods. As shown in the table, SAX^{++} reached p-values that were less than 0.01 (p-value < 0.01), which means that the SAX^{++} representation is a highly significant representation compared to that obtained using previous methods. This result indicates that this method always tries to achieve the maximum p-values compared to SAX. Small p-values provide evidence against the null hypothesis because the observed data are improbable. As shown in Table 4, the statistical analysis of each dataset for HS^{MPAR}-*RF* with SAX revealed that the p-value was less than 0.01, which supports the efficiency of the proposed method.

Columns 3 and 8 of Table 5 show the optimal alphabet sizes that were generated by the SAX^{++}algorithm. These alphabet sizes were chosen as the optimal alphabet after the HS algorithm ran 10 times for each dataset. The SAX^{++}algorithm performed better in 16 datasets in terms of maximising information (*alphabet size*) when compared to the *alphabet sizes* produced by GENEBLA (a-G) and the original SAX (a-SAX). This result is demonstrated in Table 5, whereas SAX^{++} performed as well as the original SAX and GENEBLA algorithms in the datasets faceAll and wafer, which are also shown in the table.

Table 5. The Alphabet Size for SAX++, GENEBLA and Classical SAX.

No	Data sets	a- SAX^{++}	a-G	a-SAX
1	coffee	15	5	10
2	Adiac	52	35	10
3	Gun Point	32	5	10
4	ECG200	50	11	10
5	Beef	20	17	10
6	Olive Oil	22	17	10
7	Fish	52	10	10
8	Trace	50	13	10
9	FaceFour	23	3	10
10	50word	20	6	10
11	faceAll	13	7	10
12	Swedish Leaf	30	8	10
13	Two pattern	15	4	10
14	wafer	40	40	10
15	Yoga	25	8	10
16	Light7	45	8	10
17	Light2	15	5	10
18	Osu Leaf	25	4	10
19	Control chart	10	6	10
20	CBF	10	3	10

SAX^{++} produced 10 alphabet sizes, and the original SAX also produced 10 alphabet sizes for the same data (faceAll dataset). For the wafer dataset, the SAX^{++} algorithm produced 40 alphabet sizes. The SAX^{++} algorithm performed worse than the original SAX in only two data sets (Control chart and CBF). When compared to SAX in control chart dataset, the SAX^{++} algorithm produced 8 alphabets, whereas SAX yielded 10 alphabets for same data. The SAX^{++} algorithm also performed worse with the CBF dataset; the SAX^{++} algorithm yielded just 7 alphabets, whereas SAX yielded 10 alphabets. This result is shown in Table 5. Figure 2 shows the difference between SAX^{++}, GENABLA and SAX in term of the alphabet size defined by the SAX process.

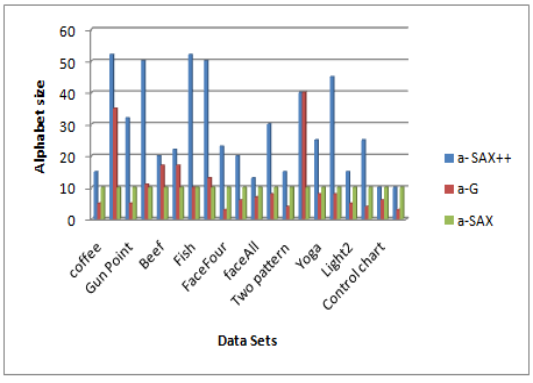

Fig 2: Alphabet size for SAX^{++}, GENEBLA and Classical SAX.

The word sizes that were produced by SAX^{++}, GENEBLA and SAX are denoted in Table 6 as *w*-SAX^{++}, *w*-G and *w*-SAX, respectively. SAX^{++} yielded large generated word sizes in 16 data sets. GENEBLA generated large word sizes in only 2 datasets, as did SAX.

Table 6. The word sizes for SAX^{++}, GENEBLA and Classical SAX.

No	Data sets	TS	w-SAX^{++}	w-G	w-SAX
1	coffee	286	166	93	128
2	Adiac	176	102	89	64
3	Gun Point	150	86	61	64
4	ECG200	96	49	41	32
5	Beef	470	236	233	128
6	Olive Oil	570	189	231	256
7	Fish	463	187	134	128
8	Trace	275	141	126	128
9	FaceFour	350	139	106	128
10	50word	270	140	89	128
11	faceAll	131	55	68	64
12	Swedish Leaf	128	62	56	56
13	Two pattern	128	61	59	32
14	wafer	152	74	78	64
15	Yoga	426	209	176	128
16	Light7	319	119	89	128
17	Light2	637	204	155	256
18	Osu Leaf	427	234	106	128
19	Control chart	60	45	31	16
20	CBF	128	66	30	32

The comparison between SAX^{++}, GENEBLA and SAX is shown in Figure 3. This figure demonstrates the difference between each algorithm that is associated with each dataset.

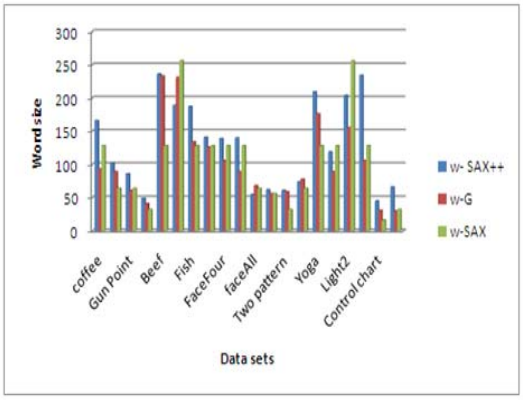

Fig. 3. Word sizes for SAX^{++}, GENEBLA and Classical SAX.

4.3 Discussion

The advantage of the proposed method is that SAX^{++} does not require a priori parameters because these parameters are automatically calculated; moreover, RF works very well whenever the time series data length is large. RF and HS^{MPAR} improved the performance of SAX, especially in datasets where SAX improved the classification performance of larger alphabet sizes. Hence, most of the datasets improved the performance of SAX when word size was defined by the parameters that were found using SAX^{++}. We decided to compare SAX^{++} to GENEBLA because GENEBLA is one of the most efficient methods that have been proposed thus far. RF allows greater control over the selection of the number of

intervals because it uses a relative frequency to compute the interval size for the data. In addition, RF uses the fundamentals of the distance algorithm to merge the confidence interval values and the lower confidence interval values. Based on the two functions used in RF, the distance measure may also contribute to achieving better classification performance. SAX, on the other hand, uses its own measure of distance to reconstruct part of the original data while maintaining the relationship between the discrete representation and the representation of the continuous time series data.

We conclude that SAX^{++} performed better than the other techniques that were used for comparison in the previous experiment. The ability to optimise the appropriate number of intervals (the alphabet size) and the number of characters in the word (the word size) ensures that important knowledge patterns are retained with good accuracy; this scenario has inspired us to test the RF and HS^{MPAR} using time series datasets to observe its effectiveness in temporal patterns. One feature in applications of most time series data is the ability to manage and control the amount of information loss, whereas some of the discretisation techniques hide important points. Hence, it is very important to retain the integrity of the data representations that are generated for a specific domain.

Experiments further verified that the performance of SAX^{++}, in which RF and HS^{MPAR} are integrated to generate the alphabet and the word sizes to be used in the SAX time series representation, was superior. Larger word and alphabet sizes and the ability to obtain lower error rates indicate that the proposed algorithm manages to preserve more knowledge by retaining most of the original data values. Given a labelled dataset that contains temporal data, the RF and HS^{MPAR} algorithms automatically compute only the alphabet size (a) and the word size (w). The efficiency of the method was evaluated using 20 datasets and was compared to some of the most efficient representation methods that are known for time series, which are called SAX, GENEBLA and raw data. Usually, error rates obtained via the SAX^{++} representation are similar to those obtained by GENEBLA tests using the same parameters (alphabet size and word size). SAX^{++} performed at an error rate (18 wins/ 2 losses/ 0 ties) that outperformed SAX (15 wins/ 5 losses/ 0 ties) and GENEBLA (15 wins/ 3 losses/ 2 ties).

V. Conclusion

We propose the integration of the RF approach and the HS^{MPAR} algorithm to improve the performance of SAX, a symbolic time series data representation. Both RF and HS^{MPAR} have a specific feature that, when integrated, provides advantages to improve the previous SAX method. RF lends its advantage in using a frequency method to generate the alphabet size of a long time series, whereas RF lends the ability to produce a large number of intervals with respect to maintain accuracy. Using the frequency function in conjunction with HS^{MPAR} with multi-pitch adjustment yields improved solutions in each iteration and HS^{MPAR} is capable of using discrete variables.

The main feature of RF is the threshold value, which is calculated based on the accumulated relative frequency of each interval. Setting the lower and upper boundaries that are based on this threshold has enabled us to ensure that a larger number of intervals is generated; thus, more information is preserved. The results have shown that the proposed SAX^{++} method potentially generates a larger number of intervals as well as more accurate alphabet and word sizes with lower error rates. When continuous data, such as time series data, must be discretised, the aim is to optimise the bin number without losing information. This goal is very important when addressing weather data, in which patterns from each time series may contribute important information. The SAX^{++} method is appealing in discretising time series data because of its simplicity. This advantage has made RF and HS competitive when integrated with other time series data representations.

References

[1] E. Keogh, S. Chu, D. Hart, M. Pazzani, An online algorithm for segmenting time series, in: Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, 2001, pp. 289-296.

[2] H.-G. Acosta-Mesa, F. Rechy-Ramírez, E. Mezura-Montes, N. Cruz-Ramírez, R. Hernández Jiménez, Application of time series discretization using evolutionary programming for classification of precancerous cervical lesions, Journal of Biomedical Informatics, (2014).

[3] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, Fast subsequence matching in time-series databases, SIGMOD Rec., 23 (1994) 419-429.

[4] K.F. Chan, A. W, Efficient Mining of Partial Periodic Patterns in Time Series Database, in: Proceedings of the 15th International Conference on Data Engineering, IEEE Computer Society, 1999, pp. 106.

[5] J. Lin, E. Keogh, S. Lonardi, B. Chiu, A symbolic representation of time series, with implications for streaming algorithms, in: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, ACM, San Diego, California, 2003, pp. 2-11.

[6] C.-G.A.-M. Daniel-Alejandro Garc, Discretization of Time Series Dataset with a Genetic Search, in: Proceedings of the 8th Mexican International Conference on Artificial Intelligence, Springer-Verlag, Guanajuato, M\&\#233;xico, 2009, pp. 201-212.

[7] r. Fabian Mörchen, Alfred Ultsch, Optimizing time series discretization for knowledge discovery, in: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, ACM, Chicago, Illinois, USA, 2005, pp. 660-665.

[8] F. Rechy-Ram, G.A. Mesa, Efr, n. Mezura-Montes, N. Cruz-Ram, Times series discretization using evolutionary programming, in: Proceedings of the 10th international conference on Artificial Intelligence: advances in Soft Computing - Volume Part II, Springer-Verlag, Puebla, Mexico, 2011, pp. 225-234.

[9] A.M. Ahmed, A.A. Bakar, A.R. Hamdan, Harmony Search algorithm for optimal word size in symbolic time series representation, in: Data Mining and Optimization (DMO), 2011 3rd Conference on, 2011, pp. 57-62.

[10] A.B. Layeb, Seriel Rayene, A Novel Quantum Inspired Cuckoo Search Algorithm for Bin Packing Problem., International Journal of Information Technology & Computer Science Vol. 4 (2012) p58-67.
[11] X. Xi, E. Keogh, C. Shelton, L. Wei, C.A. Ratanamahatana, Fast time series classification using numerosity reduction, in: Proceedings of the 23rd international conference on Machine learning, ACM, Pittsburgh, Pennsylvania, 2006, pp. 1033-1040.
[12] B.-K. Yi, C. Faloutsos, Fast Time Sequence Indexing for Arbitrary Lp Norms, in: Proceedings of the 26th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 2000, pp. 385-394.
[13] K. Chakrabarti, E. Keogh, S. Mehrotra, M. Pazzani, Locally adaptive dimensionality reduction for indexing large time series databases, ACM Trans. Database Syst., 27 (2002) 188-228.
[14] G.-L. Alexander, Discretization algorithm Time Series Based on Aplicaci'on in entropy and colposcopic data, Universidad Veracruzana, (2007).
[15] D. Garc, Acosta-Mesa, Discretization of Time Series Dataset with a Genetic Search, in: Proceedings of the 8th Mexican International Conference on Artificial Intelligence, Springer-Verlag, Guanajuato; Mxico, 2009, pp. 201-212.
[16] E. Dimitrova, McGee, J., Laubenbacher, E, Discretization of Time Series Data, eprint arXiv (2005).
[17] A.M. Ahmed, A.A. Bakar, A.R. Hamdan, Improved SAX time series data representation based on Relative Frequency and K-Nearest Neighbor Algorithm, in: Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on, 2010, pp. 1320-1325.
[18] A.A. Bakar, A.M. Ahmed, A.R. Hamdan, Discretization of time series dataset using relative frequency and K-nearest neighbor approach, in: Proceedings of the 6th international conference on Advanced data mining and applications: Part I, Springer-Verlag, Chongqing, China, 2010, pp. 193-201.
[19] J. Shieh, E. Keogh, iSAX: indexing and mining terabyte sized time series, in: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, Las Vegas, Nevada, USA, 2008, pp. 623-631.
[20] J. Lin, Y. Li, Finding Structural Similarity in Time Series Data Using Bag-of-Patterns Representation, in: Proceedings of the 21st International Conference on Scientific and Statistical Database Management, Springer-Verlag, New Orleans, LA, USA, 2009, pp. 461-477.
[21] F. M, A. Ultsch, Optimizing time series discretization for knowledge discovery, in: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, ACM, Chicago, Illinois, USA, 2005, pp. 660-665.
[22] H.G. Acosta Mesa, Nicandro, C.R., Daniel-Alejandro, G.-L, Entropy Based Linear Approximation Algorithm for Time Series Discretization, In: Advances in Artificial Intelligence and Applications, 32 (2005) 214–224.
[23] G.-L. Alexander, Algorithm discretization Time Series Based on Aplicaci'on in entropy and colposcopic data, Universidad Veracruzana (2007).
[24] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, Madison, Wisconsin, 2002, pp. 1-16.
[25] U.M.F.a.K.B. Irani, Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning, In: Proceedings of the 13th International Joint Conference on Artificial Intelligence, (1993) 1022-1029.
[26] R.J. Larsen, M.L. Marx, An Introduction to Mathematical Statistics and Its Applications, Prentice-Hall, 1986.
[27] E. Keogh, Xi, X., Wei, L., R, C.A, The UCR Time Series Classiffication/Clustering Homepage http://www.cs.ucr.edu/~eamonn/time_series_data/, (2006).

Authors' Profiles

Almahdi Mohammed Ahmed is Postdoctoral Researcher in the Center for Artificial Intelligence Technology UKM. He received his PhD degree in computer science from the University Kebangsaan Malaysia. Dr. Almahdi is interested to the Data Mining Pre-processing like discretization, reduction and representation of time series and combinatorial optimization methods and its application to solve several problems from different domains like Weather problem and electric load problems. Dr. Almahdi has many publications in Data mining and applications.

Azuraliza Abu Bakar is Prof. Dr. Head of Centre for Artificial Intelligence Technology (CAIT). She got her BSc, MSc from UKM, and got her PhD from UPM Malaysia

Prof. Dr. Azuraliza is interested to the Data Mining Pre-processing like discretization, reduction and representation of time series and combinatorial optimization methods and its application to solve several problems from different domains like Weather problem and medical data problems. Prof. Dr. Azuraliza has many publications in Data mining and applications.

Abdul Razak Hamdan is Prof. Dr. Dean, Head of the Data Mining active. He got his BSc from (UKMalaysia),and MSc from (Ncle, UK), and PhD from (Lboro) Department of Management Science and Systems Faculty of Technology and Information Sciences Universiti Kebangsaan Malaysia.

Prof. Dr. Abdul Raza**k** is interested to the Data Mining Pre-processing like discretization, reduction and representation of time series and combinatorial optimization methods and its application to solve several problems from different domains like Weather problem and medical data problems. Prof. Dr. Abdul Razak has many publications in Data mining and applications.