ORIGINAL RESEARCH

# Encoding seasonal patterns using the Kasai algorithm

Kenneth M. M'Balé*, Darsana Josyula

*Bowie State University, United States*

**Received:** March 2, 2017 **Accepted:** May 31, 2017 **Online Published:** June 14, 2017
**DOI:** 10.5430/air.v6n2p80 **URL:** https://doi.org/10.5430/air.v6n2p80

## ABSTRACT

Data series that contain patterns are the expression of a set of rules that specify the pattern. In cases where the data series is known but the rules are not known, the Kasai algorithm can analyze the data pattern and produce the complete set of rules that described the data pattern observed to date.

**Key Words:** Seasonality, Seasonal patterns, Data patterns, Time series, Grammar, Rules

## 1. INTRODUCTION

Data points under study can be examined individually or in a group. A datum carries its significance individually within its own context. Data in a group carries significance both at the datum and in terms of the datum's participation in the group. In this paper, we focus on data in group, specifically, an ordered group where the position of the datum in the sequence is also significant. This type of data group is called a data series. Our objective is to detect patterns of elements in a data series. For example, imagine we are observing the behavior of an individual over a week. Each element in the data series represents a behavior such as [eat food], [greet friend], [go to work], and so on. We want to identify a pattern such as [brush teeth] always happens after [wake up]. The elements of the series are symbols; there is no mathematical relationship between them, there is only an order of occurrence.

The concept of order allows us to distinguish between two types of series, random and systematic. A random data series contains no patterns in the occurrence of datum within it. For example, a data series such as P1 = [k, b, a, z, q, p, m] is random because it contains no pattern. On the other hand, a data series such as P2 = [k, a, b, k, a, b] is systematic as it contains a pattern. The Kasai algorithm recognizes patterns in systematic data series.

There are several types of systematic patterns in the environment. For example, we can consider weather. In the temperate zone of Earth, there are four annual seasons; spring, summer, fall and winter. These seasons repeat in the same cycle. Within each season, there are also sub-seasons. For example, in North America, we experience an Indian Summer in the fall. Consider the climate over 50,000 years. From this perspective, we observe similar cycle of warming and cooling. For example, ice ages last several thousand years, followed by warming period of several thousand years. We can also refer to an ice age as a season that possesses subseasons.

An Epoch is the period in the pattern that it takes for it to cycle. For example, the epoch is the year when we consider the four seasons. When we consider the ice ages, the epoch is 50,000 years. The annual epoch is a sub-epoch of this longer epoch.

The detection of a pattern in a data series is a function of the method used to detect it. For example, P1 does not have a pattern in terms of its components. However, there is a

*Correspondence: Kenneth M. M'Balé; Email: kmbale@gmail.com; Address: Bowie State University, United States.

pattern when one considers the relationships between [k, b, a] in the alphabet and notices that the relationship is the same as [z, q, p]. This relationship lets us predict what should follow [m] if P1 is systematic. The pattern in P2 is obvious and doesn't depend on the lexical relationships of the letters. Another way to make this observation is that detection methods cannot detect patterns they are not designed to detect. The elements of the data pattern are symbols. We do not assume there is an embedded relationship between them. Our use of letters, for example, is not meant to imply a lexical ordering. In this context, every element of the data series must be considered. If the element can be detected, it is not noise.


**Figure 1.** Reflexive pattern

When we consider the types of systematic patterns that can be constructed, we identify two types of series we call reflexive (see Figure 1) and periodic (see Figure 2). A reflexive pattern uses the same symbol; P3 = [a, a, a, a, $\cdots$ ]. A periodic pattern repeats a series of symbols; P4 = [a, b, c, a, b, c, $\cdots$ ].

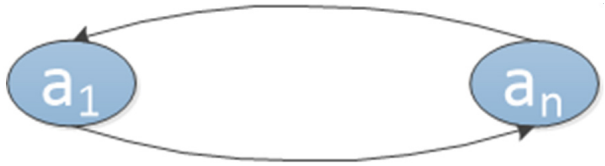


**Figure 2.** Periodic pattern

In the figures, the edge with arrow indicates that a sequence repeats at some point. In the case of P3, it repeats immediately. In the case of P4, it repeats after the symbol [c], when n = 3.

The next type of systematic pattern is composed of subsequences of symbols. We denote a subsequence using a capital letter. A subsequence is a finite symbol series such as S = [a, b, c]. A cyclical pattern occurs when a sequence periodically occurs in the series. Consider the series P5 = [a, b, c, a, b, c, a, b, k, a, b, c, a, b, c, a, b, k, $\cdots$ ]. It contains subsequences S = [a, b, c] and R = [a, b, k]. A subsequence could be shown as P5 = [S, S, R, S, S, R, $\cdots$ ] reducing it to a periodic pattern. Cyclical patterns are periodic patterns of subsequences.

A cyclical pattern contains a cycle, which the repetition of one of more subsequences. For example, in series P5, subsequence S cycles once before subsequence R occurs. At the symbol level, there are two occurrences of symbol [b] before symbol [k] occurs. This edges capture this behavior in Figure 3. In practice, edges are labelled with the cycle count as in Figure 4.

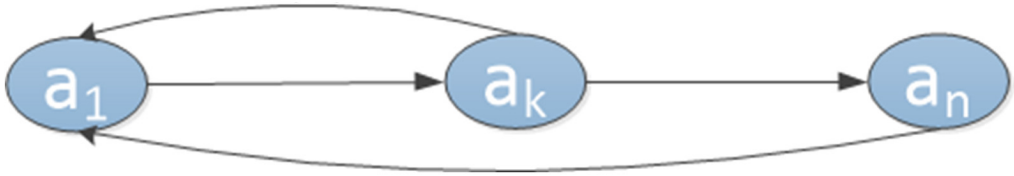


**Figure 3.** Cyclical pattern

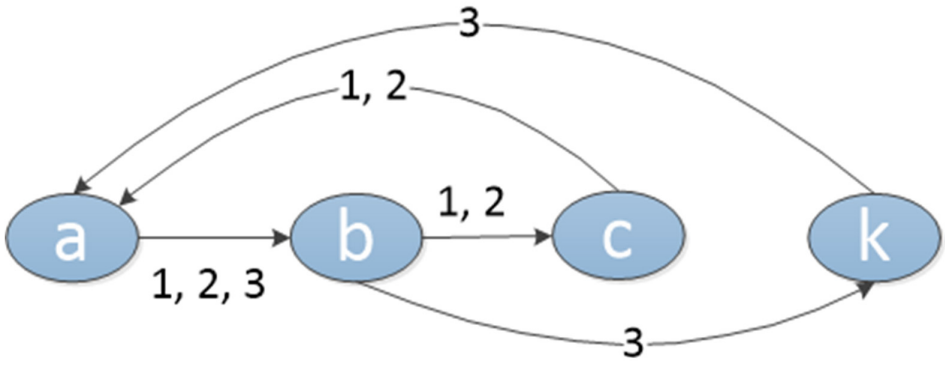


**Figure 4.** Labelled cyclical

The graph in Figure 4 produces [a, b, c] on cycles 1 and 2, and [a, b, k] on cycle 3, ad infinitum, just like series P5. We can visualize each sequence as a season, and a series of sequences as an epoch. The seasons are occurring in a certain order over an epoch. An epoch is the period over which all sequences appear at least once, before starting again. Within each season, there can be sub-season or sub-sub-season and so on. A complex season has sub-seasons while a simple season does not. This observation leads us to define a hybrid pattern (see Figure 5) as one that contains any combination of reflexive, periodic and cyclical patterns (The letters on the edges denote the cycle count).

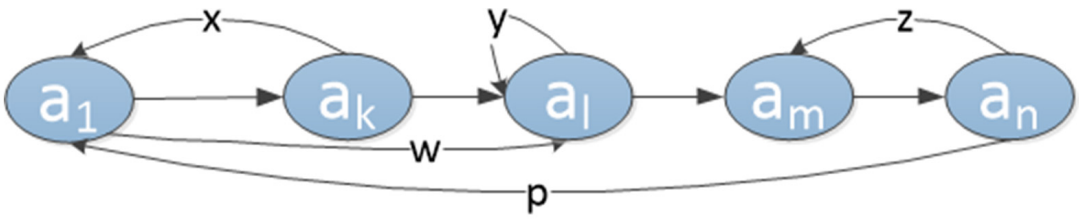


**Figure 5.** Hybrid pattern

The Kasai algorithm is a technique that processes a data series and derives the rules that produce the data series. A set of rules that mirrors a data series has several advantages. It acts as a memory because it captures the static and dynamic characteristics of the data series. It enables the prediction of future state of the data series based on the current state. It supports comparison of data series using set operations and graph analysis techniques which can be more efficient and insightful than brute force comparisons. These advantages become increasingly important as we face the data explosion of the Internet of Things.

## 2. Materials Studied

Finite state machines can be classified as deterministic and non-deterministic.[1] Deterministic machines produce one trace for any given input while non-deterministic machines can produce multiple traces. Probabilistic state machines have a probability assigned to each transition. The Kasai is similar to a deterministic state machine in the sense that it can reproduce one and only one input sequence. Similar to the probabilistic state machine, its transitions carry attributes that affect the selection of the transition based on the state of the input.

Using VIATRA2 and Petri nets,[2] the RETE approach is used for incremental graph pattern matching. The authors apply the language VIATRA2 and demonstrate their approach using Petri nets. In VTCL, graph transformation rules are specified using a precondition on the left-hand side and the postcondition on the right-hand side. Similarly, the Kasai defines the rules that describe the input sequence as a precondition implying a postcondition.

Using ILOG JRules,[3] and the RETE algorithm, a user has to specify the rules based on their knowledge of the event domain the application will process. A Kasai object can detect rules in the event stream and either present them to the person for approval or automatically specify the rules and update the rules engine. The approach for augmenting RETE described in Schor *et al.*[4] suggests mechanisms that can be used to combine the Kasai with RETE.

The traditional RETE algorithm does not support temporal operators. Several extensions have been proposed to enable complex event processing using RETE.[5,6] The Kasai object natively supports a representation of time. This representation of time is relative to itself. The charge built through cycle traversal describes the temporal constraints inherent within the input sequence. Timing is part of the description of the rules the Kasai generates.

Intrusion detection continues to be significant problem.[7] Detection approaches can be categorized as anomaly detection or misuse detection. Anomaly detection assumes that intrusive activity varies from a norm. Anomaly detection relies on establishing a statistical model and detecting large variances. Misuse detection focuses on behavior and detecting unusual patterns. The authors describe a misuse detection approach based on state transition analysis by using pattern matching to detect system attacks. The Kasai object encapsulate the signature layer and the matching engine into a single object.

## 3. Methods

The Kasai dynamically builds a set of rules that describe the sequence processed to date. A Rule takes the form subsequence $\rightarrow$ symbol. The rule $S_x \rightarrow t_n$ denotes that subsequence $S_x$ predicts symbol tn. Within the Kasai, the collection of rules is represented as a directed graph. The nodes of the graph are the rules. The edges are directed and for a unique path through the nodes. The graph is fully connected and all nodes are reachable. The first rule added to the graph is referred to as the Root Rule.

A Path is an ordering of edges that leads back to the root rule. Since all rules are connected, any node could be designated as the root. By convention, the first rule discovered is the root. However, the best root is the most frequently occurring rule in the order. Unfortunately, the most frequently occurring rule may not be known at the outset, or, it can change over time. The Kasai can refactor the graph to reposition the root rule.

The collection of rule is static but the description of the ordering of the edges must include the dynamic aspects of the series as well. To capture the dynamic aspects, the Kasai uses a representation of cycles. This enhanced graph, that captures both static and dynamic aspects, is called a Sarufi. Figure 4 is an example of a Sarufi. A pattern is seasonal whenever its Sarufi has cycles greater than one (1).

Each cycle in a Sarufi has a charge. As we traverse the Sarufi, the charge builds by one each time through the root node. When the charge reaches the cycle value, the cycle is active. Once the cycle is traversed, its charge goes back to zero (0). We define an Ideal Sarufi as one when the root node is in cycle 1 and there is a path from each node to any other node. For example, a Sarufi of a genome will be ideal. However, the Sarufi of weather will not be ideal immediately because it starts somewhere in the middle of the weather pattern. Eventually, non-ideal Sarufi will become ideal because the algorithm refactors the Sarufi as it discovers new patterns in the data. Systematic patterns result in ideal Sarufi. Random pattern cannot. When faced with a non-ideal Sarufi, one can conclude that the input series is random once the Kasai has processed the complete series.

The Kasai algorithm processes the input sequence by testing the existing set of rules to determine if they predict the current input. If the prediction is correct, it predicts the next input. If the prediction is incorrect, it revises the rules. The algorithm, as described below, is always updating and learning. It is straightforward to disable the learning mode by modifying the third ELSE clause of the MAINLINE loop.

The algorithm listed is not an efficient implementation. The practical implementation uses a multithreaded implementation with slightly different logic. The multithreaded implementation is more complex but it produces the same results. The Kasai algorithm is described in Algorithm 1.

**Algorithm 1** The Kasai algorithm

```
Globals:
        activeRule = null
        predictedsymbol = null
        cycleOrder = 0
        firstsymbol = read()
Mainline:
        loop:
                _symbol = read()
                if _predictedsymbol == null
                        addRule(pwGetState(), _symbol)
                        predict(pwGetState())
                else if predictedsymbol = _symbol
                        activeRule.cycleCount = activeRule.cycleCount + 1
                        activeRule.cycleOrder = cycleOrder
                        predict (pwGetState())
                else //predictedsymbol <> _symbol
                        addRule(pwGetState(), symbol)
                        predictedsymbol = null
                        activeRule = null
                end if
        until no more symbols;
end Mainline
predict(LHS):
        _psymbol = null
        _LHS = LHS - first symbol of LHS
        _rule = pwGetRule(LHS)
        while (_rule == null) and length(_LHS) > 0
                _rule = pwGetRule(_LHS)
                _LHS = _LHS - first symbol of _LHS
        end while
        if _rule != null  //rule is found
                _psymbol = _rule.RHS
                activeRule = _rule
        end if
        predictedsymbol =  _psymbol
end predict
pwGetState():
        _newLHS = null
        if firstsymbol == null
                for all rules in _rule.cycleOrder where _rule.cycleOrder > 0
                        for _rule.cycleCount
                                newLHS = newLHS +_rule.LHS
                _newLHS = _newLHS + _rule.RHS  //RHS of last rule in cycle
        else
                _newLHS = firstsymbol
                firstsymbol = null
        end if
        return _newLHS
end pwGetState
pwGetRule(LHS, RHS):
        return (_rule where _rule.LHS = LHS and _rule.RHS = RHS) or null
end pwGetRule
pwGetRule(LHS):
        return (_rule where _rule.LHS = LHS) or null
end pwGetRule
```


{[a],[a],[a],….}

**Figure 6.** Epoch with one season. An infinite series of the same season where each epoch has only one season. Kasai represents this type of series as a reflexive pattern.

## 4. DISCUSSION

In this section, we discuss the algorithm's ability to process any type of systematic patterns. We present possible applications of the algorithms.

### 4.1 Completeness

The Kasai algorithm produces only the rules reflecting the data series it has processed to date and the rules it produces

fully reproduce the data series. We can classify seasons and epochs. A simple season has no sub-seasons. A complex season contains at least one sub-season. A simple epoch has no sub-epochs. A complex epoch contains at least one sub-epochs. Referring to Figure 4, the length of an epoch is the sum of the cycle traversals the pattern contains. Figures 6-9 represents types of data series patterns the Kasai algorithm processes.

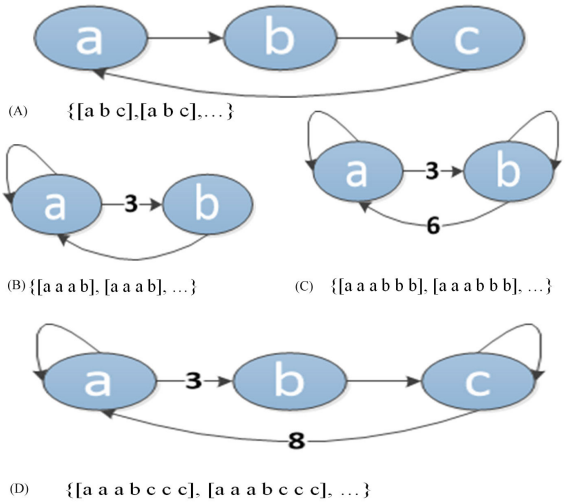


**Figure 7.** Epoch with distinct multiple season. (A) An infinite series of multiple seasons of the same length. In this example, each epoch has 3 seasons [a], [b] and [c]. This case corresponds to a periodic pattern in Kasai. (B) An infinite series of multiple seasons of different lengths. This is obtained by combining a finite number of case 1 epochs with a simple season. In this example, each epoch has 3 [a] seasons followed by a [b] season. This case corresponds to a reflexive and periodic pattern in Kasai. (C) An infinite series of multiple seasons of different lengths. This is obtained by combining a finite number of a case 1 epochs with a finite number of another case 1 epochs. In this example, each epoch has three [a] seasons followed by three [b] seasons. This case corresponds to a reflexive and periodic pattern in Kasai. (D) An infinite series of multiple seasons of different lengths. This is obtained by combining multiple finite case 1's with a simple season. This case corresponds to a reflexive, periodic and cyclical pattern in Kasai.

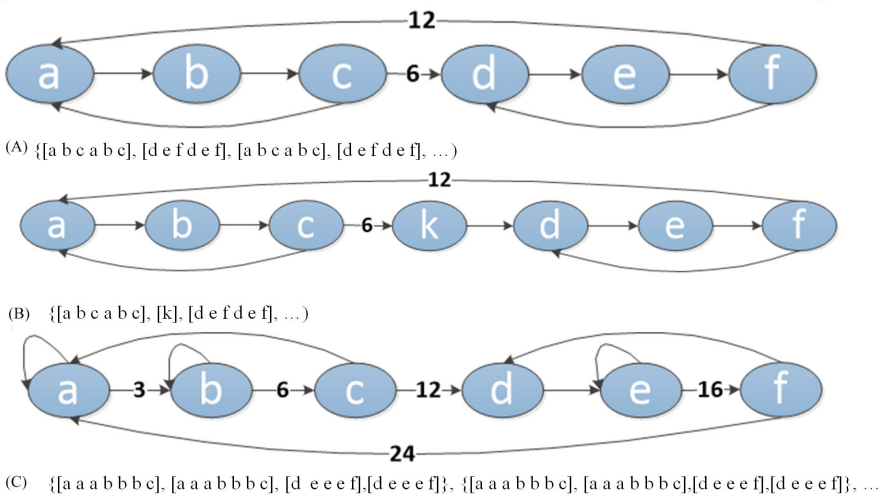


**Figure 8.** Non-overlapping complex season. (A) An infinite series of complex seasons obtained by combining multiple finite numbers of case 2 epochs. Each epoch in this example contains multiple complex seasons [a b c a b c], [d e f d e f]. This case corresponds to a periodic and cyclical pattern in Kasai. (B) An infinite series of complex seasons obtained by combining multiple finite numbers of case 2 epochs with simple seasons. Multiple complex seasons [a b c a b c], and [d e f d e f], and simple season [k]. This case corresponds to a periodic and cyclical pattern in Kasai. (C)An infinite series of complex seasons obtained by combining multiple finite numbers of case 2 epochs with simple seasons, with repeating seasons. This case corresponds to a reflexive, periodic and cyclical pattern in Kasai.

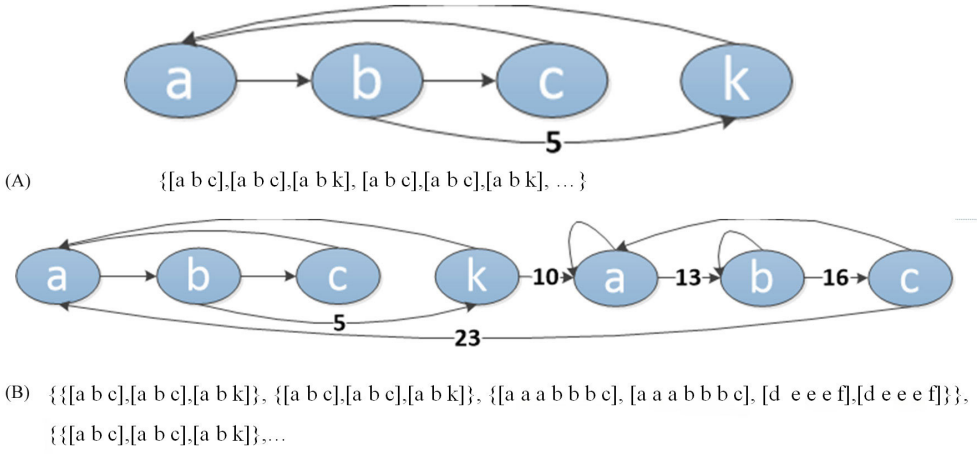


**Figure 9.** Overlapping complex season. (A) An infinite series of overlapping complex seasons. In this example, the complex season [a b c] and and the complex season [a b k] have 2 seasons that overlap ( a and b). This case corresponds to a periodic and cyclical pattern in Kasai. (B) An infinite series of overlapping and/or non-overlapping complex seasons over multiple epochs. This is obtained by combining multiple finite numbers of case 4 epochs with simple seasons or case 1 or 2 or 3 epochs. This case corresponds to a reflexive, periodic and cyclical pattern in Kasai.

Since the final case is overlapping and recursive, no sequence can be formed at a level higher than complex seasons. From this point on, we find that more complex combinations of seasons are equivalent to the final case. We conclude that seasonality in a data series can only contain reflexive, periodic, cyclical and hybrid patterns. Since the Kasai algorithm creates rules for any reflexive, periodic, cyclical or hybrid patterns, it also creates a complete set of rules for seasonal patterns.

### 4.2 Applications

We consider three types of applications for the Kasai; singleton, network and engine. A singleton is an application that uses a single Kasai object to manage on Sarufi. A network organizes a collection of Kasai objects such that the outputs of some Kasai are the inputs of other Kasai. An engine is an that combines the Sarufi by direct inspection and manipulation.

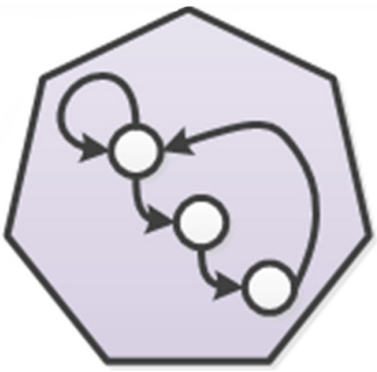

**Figure 10.** Kasai singleton sequence

#### 4.2.1 *Singleton*

A singleton Kasai (see Figure 10) produces and manage a single Sarufi that represents the input sequence it has processed to date. There are three Kasai implementation models; static, dynamic, and managed. The models refer to the way the Sarufi is updated.

A Static Kasai is trained and used to validate sequences. The Sarufi does not change in response to sequence. The Kasai only reports anomalies within the sequence as compared to the static Sarufi. An example for this model is genome analysis. In this application, we train the Kasai using a reference human genome. We can then compare other genomes or aberrant genomes to classify or to find differences.

A Dynamic Kasai immediately changes the Sarufi to reflect the patterns in the sequence. An example for this model is smart cars. A smart car needs to adjust its expectations based on changing conditions in the environment and on the road. What was normal some time ago is now anomalous because of, for example, changes in weather conditions.

A Managed Kasai is a dynamic Kasai under the control of the client application. The Kasai operates in static mode until the client application instructs it to operate in a dynamic mode. The Kasai algorithm is part of the General Purpose Metacognition Engine (GPME).[8–10] The GPME is an AI agent that enhances the performance of intelligent systems. The GPME accepts a time-series of observations from sensors. Sensory input is noisy. Therefore, the GPME creates episodes of observations. It clusters similar episodes and generates a cluster centroid episode called a Case. The cases are the inputs into the Kasai. The Kasai supplies predictions of the future state

of the environment. The GPME analyzes the anomalies to determine when the Sarufi should be modified.

A client application can apply the Kasai algorithm in several broad ways; classification, prediction, memory, and training.

Classification allows the client to analyze a sequence using a Sarufi to determine if the sequence belongs to the same class as the original sequence. A related classification is to produce the Sarufi for various sequences and compare their Sarufi.

A practical example is intrusion detection. Intrusion detection systems rely on rules to detect normal behavior. The vendor of the intrusion detection solution uses statistical analysis to develop typical profiles of behavior for the customers. Using the Kasai algorithm, each intrusion detection implementation can develop its own set of rules that more accurately reflects normal and abnormal behavior.

Prediction allows the client to determine the next valid state given all prior states. In this application, the sequence tends to be a time-series and the Kasai predicts the future state of the time-series.

A practical example is stock market prediction. We can design a symbol that consists of economic and demographic indicators, and the price of a commodity we are interested in. Like the GPME example above, some data preparation is necessary to eliminate noise and to present the data to the Kasai in a useful form. For example, we might not use actual values at market close but instead a sequence that denotes a trend or direction (Up, Down, No change, etc.). We then supply the indicators and receive the predicted value trend. In general, where the input domain is very broad, some preprocessing of the input creates a level of abstraction that simplifies the Sarufi without losing fidelity.

Memory allows the client to reproduce the original input sequence exactly. In this application, the Kasai is used to compress a large non-random data sequence into a more portable form. A practical example is genome data compression. Genome data sets contain millions of genes in the order they are found in the cell. A Kasai trained on genome eliminates redundant sequences in the genome while maintaining the fidelity of the gene sequences.

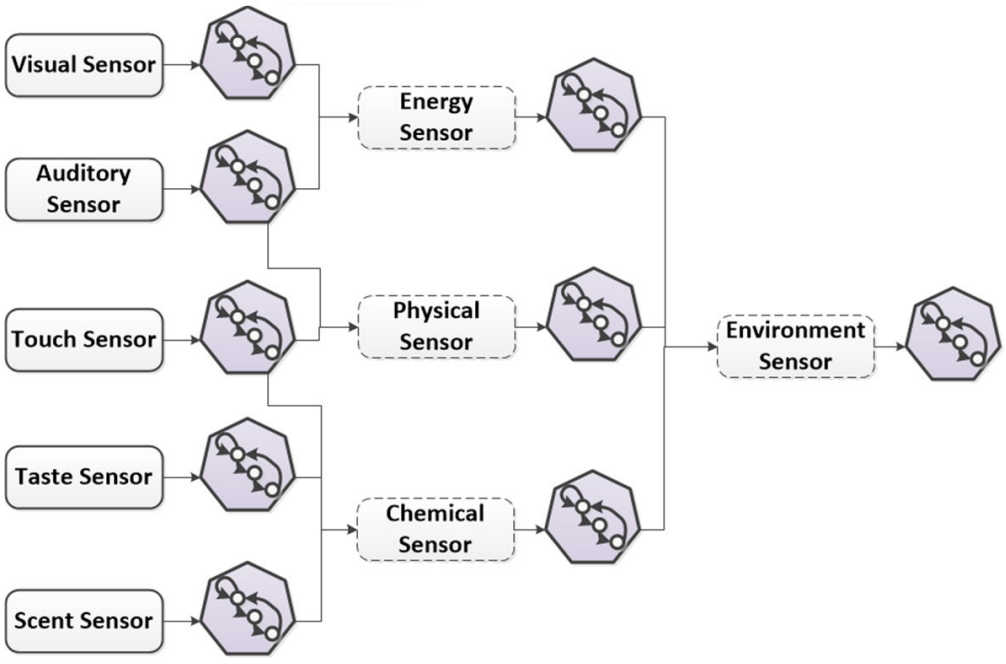


**Figure 11.** Kasai network

Training allows the client to make dynamic objects that are not naturally dynamic. In this application, the Kasai is used to train the other object. For example:

- The Rete algorithm is a pattern matching algorithm for implementing production rule systems. An implementation of a rules engine fires a rule when it database indicates that the conditions are met. It is necessary for a human designer to specify the rules to the rules engine. The Kasai algorithm can be used to identify the rules that should be implemented in the rules engine.
- An artificial neural network consists of several interconnected artificial neurons that work in unison to

solve a problem. An ANN must be trained through exposure and tuning using a process called supervised training. During training, the neural network is exposed to inputs. The designer adjusts the behavior of the neural network until it produces the correct results. The Kasai algorithm can be used to train the neural network by observing its inputs and adjusting the neural network dynamically when its outputs are incorrect.

#### 4.2.2 *Network*

A Kasai network is an arrangement of Kasai such that the output of one Kasai is the input of another. A prediction Kasai produces a prediction of the next symbol in the sequence based on the sequence processed to date.

The example Kasai network depiction in Figure 11 shows the construction of a unified environment Kasai created from the combination of other Kasai. On the left, physical sensors produce sequences that are input into their own assigned Kasai. The outputs of these Kasai are combined to form virtual sensors. In the example, the combined visual and auditory Kasai output form a virtual energy sensor. The combined auditory and touch Kasai output form a virtual physical sensor. The combined touch, taste and scent Kasai output form a virtual chemical sensor. In turn, the combined energy, physical and chemical Kasai output form a virtual environment sensor. This Kasai network enables prediction of the state of the environment. This example is like the application of the Kasai in the GPME.

#### 4.2.3 *Engine*

A Kasai Engine uses a Kasai singleton or network to produce a baseline set of Sarufi. It then manipulates the Sarufi to produce new Sarufi. The new Sarufi is the result of operations on the set of paths defined in the Sarufi.

For example, assume that we have a patient population with a medical condition we believe is genetic. We also have a population of individuals without the medical condition. We wish to know which genes contribute to the condition. Currently, this type of analysis is performed by analyzing the genomes. It can be done using a Kasai Engine.

We assign each patient genome to a Kasai resulting in a Sarufi for each genome. We perform an intersection operation on all of the Sarufi. The resulting Sarufi ($S^c$) contains the genome sequence rules for the condition as well as rules that represent gene sequences the population shares. We perform the same exercise on the individuals without the conditions and produce Sarufi Sp of healthy individuals. We now take the complement of $S^p$ and $S^c$ and produce a Sarufi $S^d = (S^c - S^p)$. Sarufi $S^d$ contains the genome sequence rules for the genes that contribute to the condition.

Sarufi calculus include all set operations since a Sarufi is a set of paths. The functions include intersection, union, subset (superset), proper subset (proper superset), not subset, power set, equality, complement (relative not absolute), difference, membership, cardinality, and empty set.

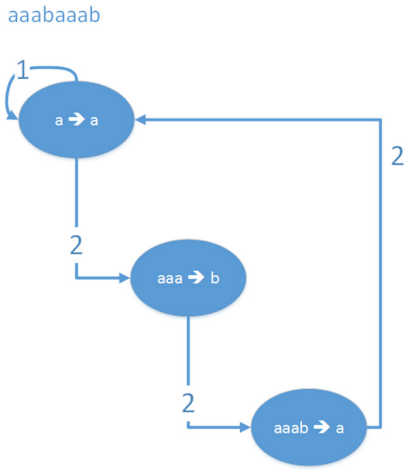


**Figure 12.** Simple reflexive seasonal pattern

## 5. RESULTS

### 5.1 Systematic data series

The next figures depict the Sarufi generated for the input sequences, in graphical form. Figure 12 depict the graphical form while the set form is: (a → a), (aaa → b), (aaab → a). The number on the edge is the number of the cycle through the root node. For example, in Figure 12, the root is (a → a). Node (aaa → b) is valid after the second pass through the root node. In Figure 14, the outermost cycle is valid after the twelfth pass through the root node.

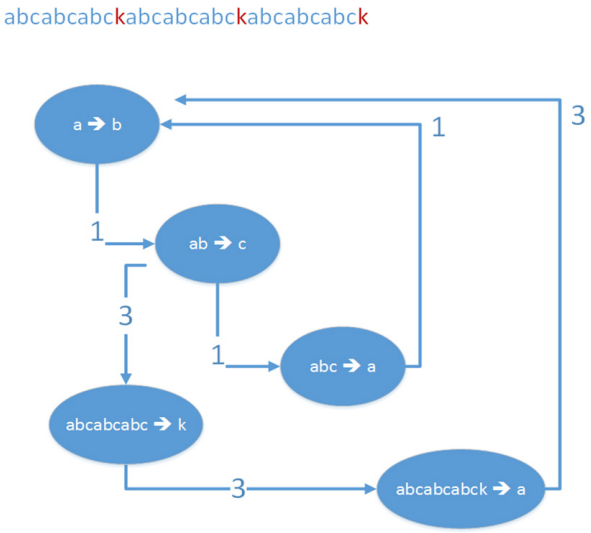


**Figure 13.** Simple seasonal pattern

Each cycle requires a certain charge built up by the traversal through the earlier cycles. Each cycle's charge is independent of others charges. In Figure 13, there are two cycles labelled with their respective charge requirements (1 and 3). After the third traversal through the root node, a charge of 3 is built up that allows travel through cycle 3. Once the root node is reached, the charge resets.

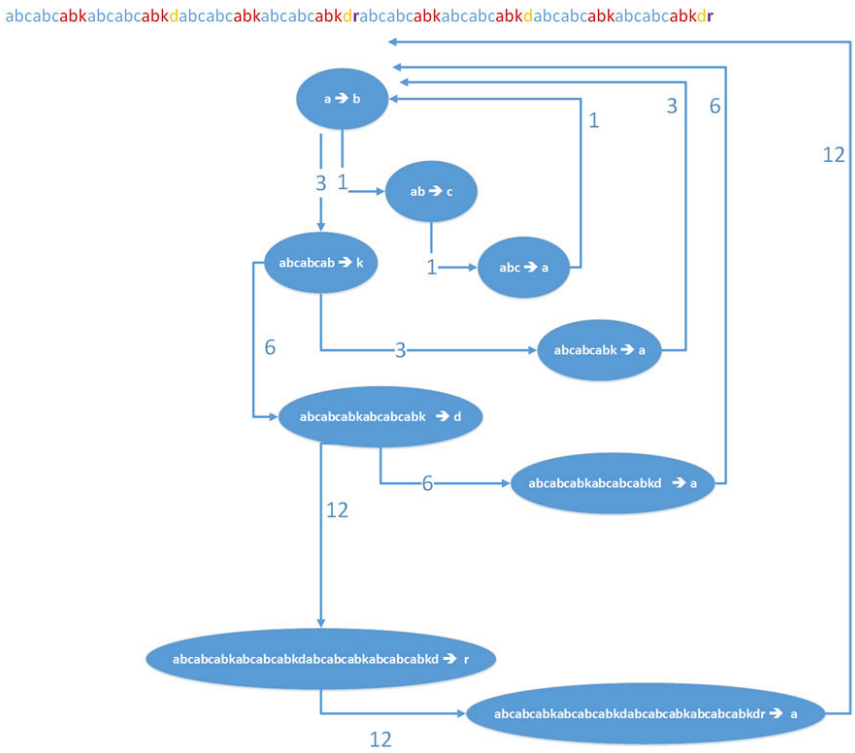


**Figure 14.** Complex seasonal pattern with multiple seasons

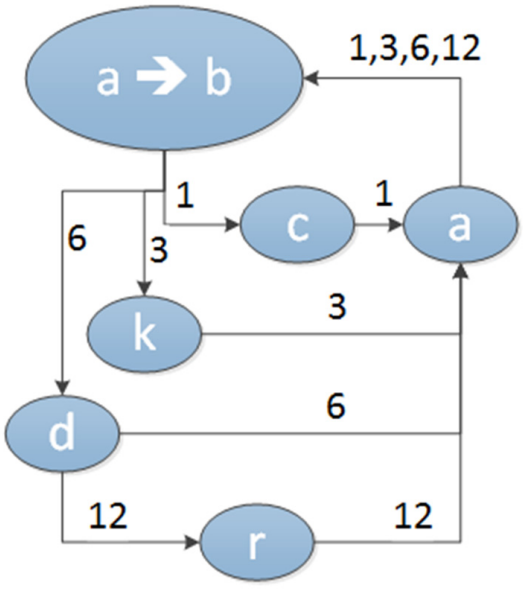


**Figure 15.** Compact representation of a Sarufi

In Figure 14, there are four cycles (1, 3, 6 and 12). Each charge builds independently and resets when the root node is reached.

The sequence contains several patterns; abc, abcabcabk, abcabcabkabcabcabkd, and abcabcabkabcabcabkdabcabcabkabcabcabkdr. The Sarufi contains a cycle for each one (1, 3, 6, 12). The Sarufi is a very compact way to represent information contained in a very long input sequence. If the sequence contains a pattern and is not random, the Sarufi contains cycles. Otherwise, at least one node is a dead end. It matters, therefore, whether the input sequence is known to be complete. For example, a genome is complete in the sense that its beginning and ending are known. Other data series may not be known to be complete.

Earlier, we described the application of the Kasai as a memory. The representation of Figure 14 can be simplified to the one shown in Figure 15.

### 5.2 Chaotic data series

As described above, the Kasai algorithm is designed to produce rules representing patterns found in systematic time series of symbolic data. Unlike numeric data, there is no mathematical relationship between elements of the data series and the Kasai algorithm does not assume the existence of such a relationship.


ISSN 1927-6974 E-ISSN 1927-6982

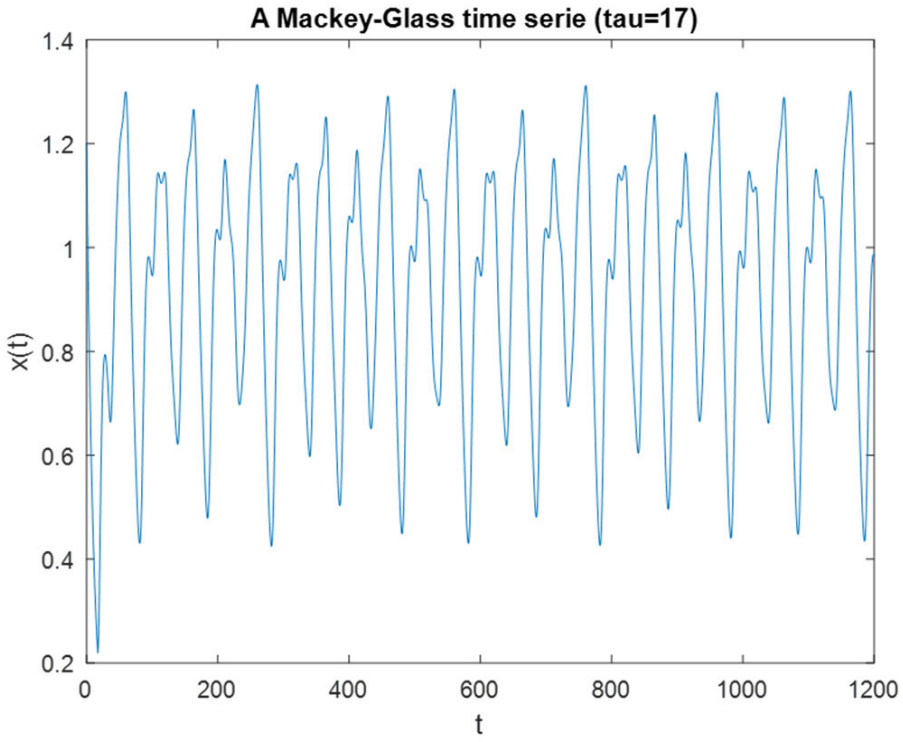


**Figure 16.** Chaotic time series

**Table 1.** A fragment of the Mackey-Glass data depicted in Figure 16

| t | x(t) |
|---|---|
| 6931 | 0.868810 |
| 6932 | 0.868810 |
| 6933 | 0.868810 |
| 6934 | 0.885138 |
| 6935 | 0.890401 |
| 6936 | 0.895573 |
| 6937 | 0.900653 |
| 6938 | 0.905640 |
| 6939 | 0.910534 |
| 6940 | 0.915333 |
| 6941 | 0.920038 |
| 6942 | 0.924647 |
| 6943 | 0.929160 |
| 6944 | 0.933577 |
| 6945 | 0.937898 |
| 6946 | 0.942122 |
| 6947 | 0.946248 |
| 6948 | 0.950278 |

Chaotic time series are technically outside the field of use of the Kasai algorithm. A random time series, where no element occurs more than once, creates a Sarufi without cycles; a set of single use Kasi. For example, a Kasi abc $\rightarrow$ d never fires because [a b c] will not happen again. In addition, the global charge remains forever at 1 because the root Kasi is never revisited since there are no cycle. Nonetheless, it is possible to use the Kasai algorithm with random time series by applying a preprocessing strategy that generates a symbol that represents the existing pattern in the time series.

The preprocessing strategy leverages patterns known to exist in the time series. Consider the Mackey-Glass differential equation chaotic time series depicted in Figure 16, using 12,000 samples, a delay factor of 17 and a time-step of 0.1 (Generated using Matlab). The resulting time series consists of 12,000 nonrepeating numbers. However, the graph shows there is a clear pattern that results from the mathematical relationship created by the formula.

Because the numbers, x(t), do not repeat, the time series is chaotic and is not a candidate for use with the Kasai directly. However, we can take the time series and transform it into a time series of symbols. Our approach is to create a bucket labelled by a symbol. For example, assume a bucket size of 3 and the time series S = [1, 2, 3, 4, 5, 6, 7, 8, 9]. We can create three buckets a = [1, 2, 3], b = [4, 5, 6] and c = [7, 8, 9]. We can then express S = [a, b, c]. Examining Figure 16, we see that the time series is bound by $1.3 > x(t) > 0.2$. We can create buckets that divide this range. Instead of listing every element of the bucket, we can define the bucket in terms of its unique symbol label, an upper bound and a lower bound

value. We denote the bucket as symbol, lower bound, upper bound.

Table 1 shows a fragment of the Mackey-Glass data depicted in Figure 16.

We can subdivide the range [0.2, 1.3] into buckets denoted symbol, lower bound, upper bound:

- $\{a, 0.868810, 0.885138\}$
- $\{b, 0.890401, 0.905640\}$
- $\{c, 0.910534, 0.924647\}$
- $\{d, 0.929160, 0.942122\}$

Given these buckets, the time series fragment in Table 1 transforms into S1 = [aaaabbbbccccdddd$\cdots$]. We can now predict the next bucket that the next value will fall into. The accuracy is a function of the size of the bucket. For example, if we create a single bucket a, 0.2, 1.3, the time series is S1 = [aaa$\cdots$.] like Figure 1. This is not very useful for prediction. At the other extreme, if we create 12,000 buckets, one for each number in the time series, then S2 = [abcdefg$\cdots$] is random and contains no seasonal patterns we can use to create Kasi. In fact, the Kasai algorithm creates 12,000 Kasi with $\kappa = 1$ but that never fire because of two reasons. First, since there are no cycles, their $c$ value never increments past 0 so c is never equal to $\kappa$. Second, $\tau$ never occurs again. The series S1 above, with a bucket size of 12,000, creates a single Kasi that fires all the time. The series S2, with a bucket size of 1, produces 12,000 Kasi that never fire. The ideal bucket size is between these two values. The bucket size is, in effect, the error tolerance of the prediction.

Our experiment compares chaotic times series prediction using the Kasai algorithm, with ANFIS and NAR implemented in Matlab. All three techniques in the experiment use the same Mackey-Glass chaotic time series as input.

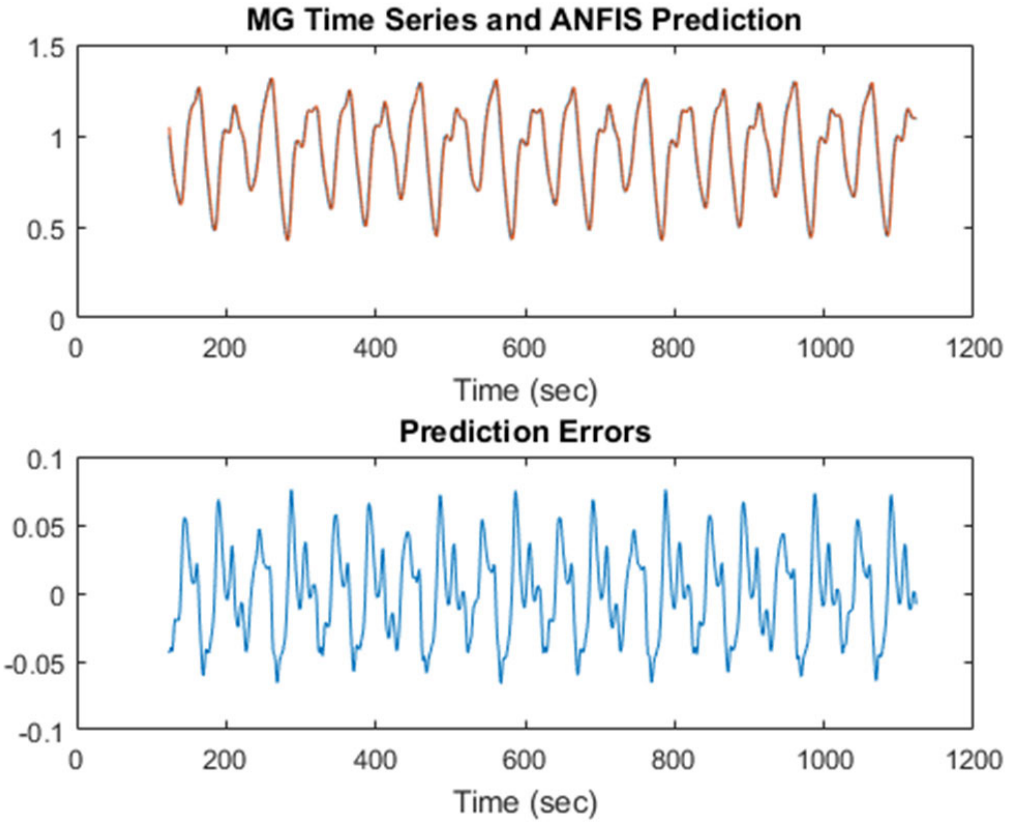


**Figure 17.** ANFIS prediction results

The ANFIS implementation can be found at: `https://www.mathworks.com/help/fuzzy/predict-chaotic-time-series-code.html?searchHighlight=mackey%20glass%20anfis&s_tid=doc_srchtitle`. The ANFIS results are shown in Figure 17. The NAR neural network implementation can be found at: `http://lab.fs.uni-lj.si/lasin/wp/IMIT_files/neural/nn05_narnet/`. The NARx results are shown in Figure 18.

The preprocessor accepts the tolerance (bucket size) as input. It dynamically creates new buckets as it processes the Mackey-Glass time series and it creates a time series of symbols that is then input to the Kasai algorithm.

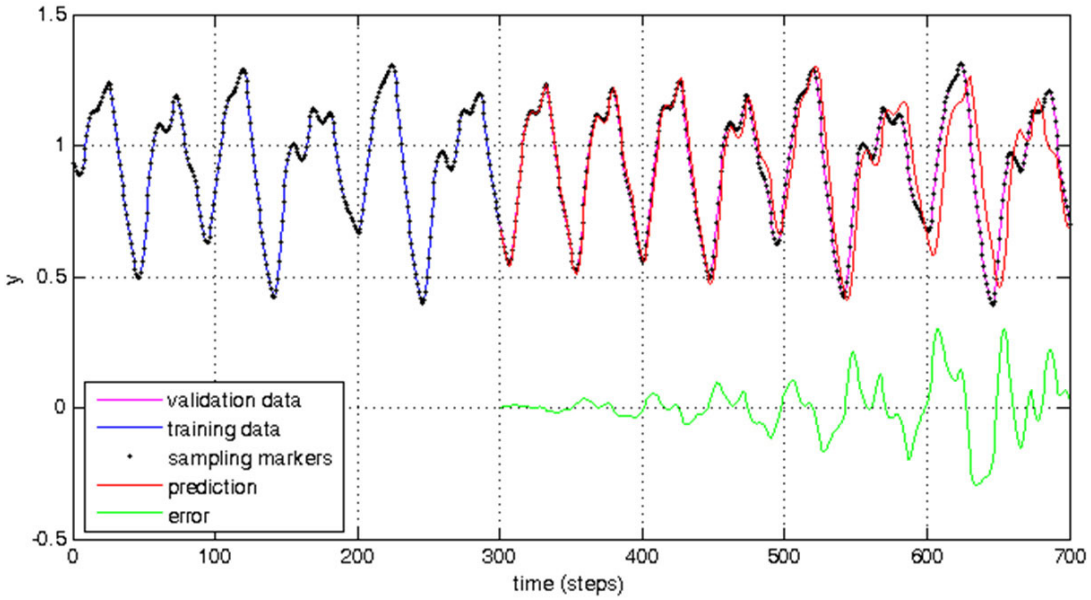


**Figure 18.** NARx prediction results

**Table 2.** The results of experiment

| Error tolerance | Predictions made | Prediction accuracy |
|---|---|---|
| 0.5 | 8017 | 67% |
| 0.1 | 6529 | 54% |
| 0.05 | 5036 | 42% |
| 0.025 | 5297 | 44% |
| 0.0125 | 5122 | 43% |
| 0.00625 | 3319 | 28% |

Table 2 shows the results of experiment. Error Tolerance is the bucket size. For example, with tolerance of 0.1, the range 0 to 1.5 divides into 15 buckets (0.0-0.1, 0.1-0.2, 0.3-0.4, $\cdots$, 1.4-1.5). Empty buckets are ignored as they are not represented in the input series to the Kasai algorithm. When the Kasai algorithm generates a prediction, the prediction is within the tolerance. Otherwise, the Kasai algorithm generates no prediction. The ideal tolerance (bucket size) is therefore a function of the problem being solved. Predictions Made is the number of predictions the Kasai algorithm makes. Prediction accuracy is the percentage of Prediction Made to the length of the data series (12,000).

The ANFIS and NAR experiments result in errors at most points of the time series. The ANFIS error range is consistent across the series. The NARX error worsens as the values gets further way from the training data set. The Kasai algorithm does not use a training data set.

Based on the results and comparing the Kasai algorithm's performance to ANFIS and NARX, it produces a reasonable prediction of chaotic time series. However, it is important to understand the context in which the Kasai algorithm was developed. It is part of an AI system that combines the symbolic procedural AI paradigm with the connectionist paradigm. For example, the ANFIS implementation uses four prior values to project a future value during the training phase of the underlying neural network. The four values are x(t-18), x(t-12), x(t-6) and x(t) to predict x(t+6). A possible better approach is to have the Kasai algorithm determine the optimal training data set. This approach should result in an improved performance for ANFIS or NAR prediction. The Kasai algorithm is not meant to compete with connectionist approaches, it is meant to enhance them.

## 6. Conclusion

In this paper, we introduce the Kasai algorithm. The Kasai algorithm analyzes an input sequence in order to generate a set of rules that describes the input sequence. The set of rules, the Sarufi, can then be used to analyze, reproduce, or compare sequences to each other, or as a memory. The rules produce an abstract and compact representation of the data series as a graph. The graph representation supports several types of operations using the rules that are difficult to implement at the discrete symbol level. Operations include comparing data series to each other and using the rules to predict future states of the data series.

## Acknowledgements

The authors wish to acknowledge Bowie State University. This work is supported by MAST Collaborative Technology Alliance – Contract No. W911NF-08-2-004 and U.S. Department of Education – Grant No. P031B090207.

## REFERENCES


[1] Cryan M. Inf1A: Probabilistic Finite State Machines and Hidden Markov Models. 2004: 44-9.

[2] Bergmann G, Ökrös A, Ráth I, et al. Incremental pattern matching in the viatra model transformation system. GRaMoT '08 Proc. third Int. Work. Graph Model Transform. 2008: 25-32. https://doi.org/10.1145/1402947.1402953

[3] Berstel B. Extending the RETE algorithm for event management. Proc. Int. Work. Temporal Represent. Reason. 2002: 49-51.

[4] Schor MI, Daly T, Lee H, et al. Advances in RETE pattern matching. National Conference on Artificial Intelligence Philadelphia. 1986: 226-32.

[5] Walzer K, Breddin T, Groch M. Relative temporal constraints in the Rete algorithm for complex event detection. International Conference on Distributed Event-based Systems. 2008: 147-55. https://doi.org/10.1145/1385989.1386008

[6] Zhou D, Fu Y, Zhong S, et al. The Rete algorithm improvement and implementation. International Conference on Information Management. 2008; 1: 426-9. https://doi.org/10.1109/iciii.2008.141

[7] Kumar S, Spafford EH. A Pattern Matching Model for Misuse Intrusion Detection. Computers & Security. 1994(2): 28.

[8] M'Balé KM, Josyula D. Handling Seasonality using Metacognition. In Cognitive 2014; 2014.

[9] M'Balé KM, Josyula D. Emulating a Brain System. In AAAI 2014 Fall Symposium. 2014: 22-7.

[10] M'Balé KM, Josyula D.Using Automatic Case Generation to Enable Advanced Behaviors in Agents. Procedia Comput. Sci. 2016; 88: 444-9. https://doi.org/10.1016/j.procs.2016.07.462