The Kasai Algorithm

K. M’Balé DSc, and D. Josyula PhD
Computer Science, Bowie State University, Bowie, MD, USA

Abstract - The Kasai Algorithm analyzes non-random data series to derive a set of rules that represents the patterns found in the data series. The rules represent a sound and compact abstraction of the data series that can be used for analysis, for reproduction of the original data series or for prediction. The Kasai Algorithm is an attempt to unify the symbolic and connectionist paradigm into a unified model.

Keywords: Pattern detection, Anomaly detection

1 Introduction

Several problems require a rule-based solution. A rule consists of a predicate and an action to be taken when the predicate evaluates true. A solution designer uses their own knowledge, experience, and goals to specify one or more static rules. Most implementations of business rules management systems rely on static rules built using variables. Dynamic rules are rules built through the observation of data.

For example, a stock trading rules system can be built with static rules governing the execution of trades. Its dynamic rules can be discovered by analyzing a time-series of stock market data and economic indicators. The combination of static and dynamic rules result in a trading system that is more adaptable than a static-only system.

Another example is a genome. Genomes consists of large non-random sequences of genes. The derivation of an equivalent set of rules can simplify the study of genomes by highlighting where genomes are different, between species or between healthy and sick individuals.

In a final example, there are several useful applications of rules management algorithm such as Rete. These applications require an analyst to develop and encode the rules into the rules engine. If it were possible to integrate such a system with a rules discovery system, these applications would become substantially more robust as they would decrease their dependency on human intervention when new patterns occur in their inputs. Thus, rules management system could be applied to a wider field of use.

In this paper, we present the Kasai Algorithm. It provides support for the definition of static rules and for the automatic discovery of dynamic rules over a data series. The Kasai encodes and tests rules discovered in a data series. It combines Bayes network and finite state machine behavior. The Kasai algorithm analyzes sequences to detect seasonal patterns within the data. The value of the algorithm is that it can detect patterns in data that human observers cannot easily find. It expresses the patterns as a set of simple rules. This set of rules can be used to reproduce the original data series or to compare data series, at an abstract level, or to predict the future state of the data series.

2 Literature Review

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 like a deterministic state machine in the sense that it can reproduce one and only one input sequence. Like the probabilistic state machine, its transitions carry attributes that affect the selection of the transition based on the state of the input.

In [2], the authors present an approach for incremental graph pattern matching using the RETE approach. 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.

In [3], the authors discuss the application of rules based programming using the RETE Algorithm. Rules are expressed in ILOG JRules. A person must 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 [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.

3 Kasai

The Kasai, is an object that provides a seasonal pattern detection service. It is embedded within the client application that accesses it through an interface.

The Kasai accepts a sequence as its input. The discrete atomic component of the sequence is called a Token. The client application supplies the tokens one at a time to the Kasai. A sequence \( S = \{t_0, t_1, \ldots, t_n\} \) of \( n \) tokens is called a Symbol. A symbol cannot be null or empty. It consists at a minimum of one token and at a maximum of the entire sequence processed to date. Since a symbol can consist of one token, every concept that applies to a token applies to a symbol, but not vice-versa. The following table defines types of sequences:

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>azbygdhsfgh</td>
<td>There are no actual patterns in the data. It is not possible to describe this pattern using any combination of grammatical rules.</td>
</tr>
<tr>
<td>Reflexive</td>
<td>aaaaaa</td>
<td>Reflexive patterns contain sequences of the same token. The grammar contains Reflexive rules such as: ( a \rightarrow a )</td>
</tr>
<tr>
<td>Periodic</td>
<td>abcababcabc</td>
<td>The pattern in the data is composed of the symbols (abc) of tokens (a, b, c). It is possible to describe this sequence by using a grammar (a set of rules). ( a \rightarrow b, ab \rightarrow c, abc \rightarrow a )</td>
</tr>
<tr>
<td>Cyclical</td>
<td>abcabcabcabcd</td>
<td>Seasonal patterns include trivial patterns. However, symbols in the sequence form patterns. The grammar for this pattern includes the trivial periodic pattern grammar above plus the rules: ( abcabc \rightarrow d, abcabcd \rightarrow a )</td>
</tr>
<tr>
<td>Hybrid</td>
<td>aaabaakaab</td>
<td>A hybrid sequence combines trivial, reflexive, or seasonal characteristics.</td>
</tr>
</tbody>
</table>

The Kasai algorithm does not perform any noise detection so the assumption is that the sequence contains no noise. Every symbol and token is considered relevant. Therefore, implementation that incorporates the Kasai may need data preparation logic prior to feeding the sequence.

The Kasai dynamically builds a set of rules that describe the sequence processed to date. A Rule takes the form symbol \( \rightarrow \) token. The rule \( S_x \rightarrow t_n \) denotes that symbol \( S_x \) predicts token \( t_n \). The collection of rules the Kasai builds is called a Grammar.

Within the Kasai, the grammar is represented as a directed graph. The nodes of the graph are the rules. The edges are directed and form 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 a sequence of edges that leads back to the starting point. Since all rules are connected, any node could be designated as the root. By convention, the first rule is the root. However, the best root is the most frequently occurring rule in the sequence. Unfortunately, the most frequently occurring rule may not be known at the outset, or, it can change over time. The Kasai can refactor the grammar to change the root rule.

The grammar is a static construct but the description of the sequence must include the dynamic aspects of the sequence as well. The grammar describes the static structure of the sequence using rules and paths. To capture the dynamic aspects, the Kasai introduces the concept of charges and cycles overlaid on top of the grammar. This graph that captures both static and dynamic aspects of the sequence is called a Sarufi. The following are examples of a Sarufi.

As the Kasai processes the sequence, it travels the paths of the grammar. We refer to this travel as a cycle because each path leads back to the rule root. The Kasai completes a cycle through the grammar each time it visits the root. Each cycle produces a charge. A cycle fires (or is activated) when its charge requirement is met. For example, in Figure 3, the cycle \( \{S_1, S_2, S_3\} \) only fires when a charge of 3 is built up. Therefore, the Kasai cycles through the Reflexive \( \{S_1\} \) twice and on the third cycle, proceeds to \( \{S_1, S_2, S_3\} \), producing the sequence \( (S_1, S_1, S_2, S_3, \ldots) \). Once a cycle fires, its charge resets to zero.
Referring to Figure 3, note that cycle 2 is implicit and not represented to keep the graph legible. Cycle 2 is identical to cycle 1 since the grammar is fully connected. The Sarufi of sequences that are purely periodic only have one cycle with charge of 1 such as Figure 2. Sarufi with cycles with charges greater than 1 have complex seasons.

The Sarufi depicted above are ideal because there is a path from each node to any other node and the root node is part of the highest frequency cycle. When the Kasai processes a sequence from the beginning, it tends to produce ideal Sarufi immediately. However, when processing begins mid sequence, the Sarufi may not be ideal for some time. For example, if a Kasai creates a Sarufi representing a genome, it will be ideal because processing starts at the beginning of the gene sequence. On the other hand, if the Kasai is processing a weather pattern, the Sarufi may not be ideal because it starts mid sequence. Eventually, non-ideal Sarufi will close and provide a path from each node to any other. Once the paths are known, the Kasai will need to refactor the Sarufi to ensure the correct node is designated as the root.

If a Sarufi does not provide a path from any node to itself through other nodes, the sequence does not repeat. In other words, there is no pattern in the sequence, it is random. Therefore, it is not possible to produce an ideal Sarufi.

4 Applications

We consider three types of applications; 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 application that combines the Sarufi by direct inspection and manipulation.

4.1 Singleton

A singleton Kasai (Figure 4) produces and manages 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.

Intrusion detection continues to be a 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 encapsulates the signature layer and the matching engine into a single object.

**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 token 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 symbol 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.

**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 producing a training data set dynamically when its outputs are incorrect.

### 4.2 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 token in the sequence based on the sequence processed to date.

The example Kasai Network depiction in Figure 5 shows the construction of a unified environment Kasai created from the combination of other Kasai.

![Figure 5 - Kasai Network](image)

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.

### 4.3 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 $S^p$ of healthy individuals. We now take the difference 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.

5 Design

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 in Table 6-1, 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.

<table>
<thead>
<tr>
<th>Table 6-1. The Kasai Algorithm</th>
</tr>
</thead>
</table>
| **Globals:**  
  activeRule = null  
  predictedToken = null  
  cycleOrder = 0  
  firstToken = read()  
  Mainline:  
  loop:  
  _token = read()  
  if predictedToken == null  
    addRule(pwGetState(), _token)  
    predict(pwGetState())  
  else if predictedToken = _token  
    activeRule.cycleCount = activeRule.cycleCount + 1  
    activeRule.cycleOrder = cycleOrder  
    predict(pwGetState())  
  else //predictedToken <> _token  
    addRule(pwGetState(), token)  
    predictedToken = null  
    activeRule = null  
  end if  
  until no more tokens;  
  end Mainline  
  predict(LHS):  
  _pToken = null  
  _LHS = LHS - first token of LHS  
  _rule = pwGetRule(LHS)  
  while (_rule == null) and length(_LHS) > 0  
    _rule = pwGetRule(_LHS)  
    _LHS = _LHS - first token of _LHS  
  end while  
  if _rule != null //rule is found  
    _pToken = _rule.RHS  
    activeRule = _rule  
  end if  
  predictedToken = _pToken  
  end predict  
  pwGetState():  
  _newLHS = null  
  if firstToken == 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  
      end for  
    end for  
    _newLHS = firstToken  
    firstToken = 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  

6 Results

The next figures depict the Sarufi generated for the input sequences, in graphical form. Figure 6 depicts the graphical form while the set form is: \{ (a \Rightarrow a), (aaa \Rightarrow b), (aaab \Rightarrow a) \}. The number on the edge is the number of the cycle through the root node. For example, on Figure 6, the root is (a \Rightarrow a). Node (aaa \Rightarrow b) is valid after the second pass through the root node. In Figure 8, the outermost cycle is valid after the twelfth pass through the root node.

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 7, 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.

Earlier, we described the application of the Kasai as a memory as shown on Figure 8. 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, abeabcabkeabcabk, and abeabcabkeabcabk. 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.

7 Conclusion

In this paper, we introduce the Kasai algorithm. The Kasai algorithm analyzes an input sequence 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.
8 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.

9 References


